RUBY / ΕΛΛΗΝΙΚΗ ΕΚΔΟΣΗ

Ruby με μικρές ασκήσεις

208 μαθήματα σύνταξης και concepts

Από τον καθημερινό κώδικα στις πιο ειδικές τεχνικές.

RUBY

```ruby
names
  .map(&:strip)
  .reject(&:empty?)
  .tap { |cleaned| audit << cleaned.length }
  .then { |cleaned| cleaned.join(", ") }
```

Ruby 3.4 / έκδοση 12.09.2026

Πριν αρχίσεις

Κάθε νέο μάθημα εισάγει μια βασική ιδέα μέσα από μικρή πρακτική δουλειά. Οι κανόνες δίνονται. Ο στόχος είναι να γράφεις και να διαβάζεις Ruby με άνεση.

Διάβασε το παράδειγμα, πρόβλεψε την επιστροφή, γράψε την προσπάθειά σου και τρέξε τα tests. Η παραλλαγή χρησιμοποιεί την ίδια γνώση σε λίγο διαφορετικό αίτημα.

Οι κύριες λύσεις βρίσκονται στο χωριστό ruby-solutions.pdf. Τα αρχεία exercise.rb είναι για τη δική σου προσπάθεια. Τα 092 και 093 ζητούν να γράψεις tests για δοσμένη υλοποίηση.

```sh
cd syntax-first/ruby
bin/lesson 014
bin/lesson 014 --solution
bin/lesson 014 --examples
```

Η πρώτη εντολή ελέγχει την προσπάθεια, η δεύτερη τη χωριστή λύση και η τρίτη τα παραδείγματα. Οι άλυτες προσπάθειες αποτυγχάνουν εσκεμμένα. Τα εργαλεία χρησιμοποιούν την ήδη εγκατεστημένη Ruby 3.4 και RSpec 3.13.2.

Οι μεγάλες γραμμές κώδικα αναδιπλώνονται οπτικά στο PDF. Το > στο περιθώριο σημαίνει συνέχεια της ίδιας γραμμής. Για ακριβή αντιγραφή κώδικα χρησιμοποίησε τα συνοδευτικά .rb αρχεία.

Περιεχόμενα

Προτεραιότητα 1: Καθημερινός κορμός

Έλεγχος δεδομένων και διαχείριση σφαλμάτων

Tests, αρχεία, JSON και μικρά εργαλεία

Προτεραιότητα 2: Χρήσιμη συνέχεια

Δικά σου blocks και callable αντικείμενα

Προτεραιότητα 3: Πιο ειδικές ανάγκες

Προτεραιότητα 4: Εξειδικευμένες τεχνικές

Πρωτόκολλα αντικειμένων και metaprogramming

Ειδική ροή εκτέλεσης και concurrency

001 / ΜΑΘΗΜΑ

Εκτέλεση και παρατήρηση · Γέφυρα

Έννοια: Ruby αρχείο, IRB, σχόλια, `puts`, `p`, `inspect` και `class`

Πριν αρχίσεις: Κανένα. Ξεκινάς εδώ.

Το αρχείο τρέχει από πάνω προς τα κάτω. Στο Terminal γράφεις `ruby example.rb`, ενώ στο `irb` δοκιμάζεις μία έκφραση τη φορά. Το `#` αρχίζει σχόλιο. Το `;` χωρίζει εντολές στην ίδια γραμμή, αν και συνήθως προτιμάμε χωριστές γραμμές. Η `puts` εμφανίζει κείμενο και επιστρέφει `nil`. Η `p` εμφανίζει την αναπαράσταση `inspect` και επιστρέφει την τιμή. Η `class` δείχνει τον τύπο. Το έτοιμο περίβλημα `def ... end` της άσκησης θα εξηγηθεί στο 004.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
price = 7
seats = 3
price * seats
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Συμπλήρωσε την έτοιμη `ticket_preview(price, seats)`: εμφάνισε με `puts` το σύνολο `price * seats` και επίστρεψε το ίδιο σύνολο. Οι δύο είσοδοι είναι μη αρνητικοί ακέραιοι. Στην κονσόλα δοκίμασε επίσης `p "21"`, `"21".inspect` και `21.class`.

RUBY
```ruby
ticket_preview(7, 3)
# => 21
```

RUBY
```ruby
ticket_preview(7, 0)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 001
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 001 --solution
```

Μια μικρή παραλλαγή

Υπολόγισε δύο εισιτήρια των 9 και επίστρεψε το αποτέλεσμα της `p`.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `18`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

002 / ΜΑΘΗΜΑ

Τιμές και εκφράσεις · Γέφυρα

Έννοια: Τοπικές μεταβλητές, αριθμητικές πράξεις και μετατροπές έγκυρων αριθμητικών κειμένων

Πριν αρχίσεις: 001

Η ανάθεση `name = expression` αποθηκεύει το αποτέλεσμα της δεξιάς πλευράς. Οι `+`, `-`, `*`, `/`, `%` είναι πράξεις. Με δύο ακεραίους το `/` κάνει ακέραια διαίρεση: `7 / 2` δίνει 3. Η `to_i` και η `to_f` μετατρέπουν έγκυρο αριθμητικό κείμενο. Εδώ τα δεδομένα δίνονται σωστά, άρα δεν μαθαίνουμε ακόμη έλεγχο εισόδου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
boxes = "4".to_i
per_box = 3
boxes * per_box + 2
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `remaining_seats(capacity_text, adults_text, children_text)`. Όλα τα κείμενα παριστάνουν μη αρνητικούς ακεραίους και η χωρητικότητα επαρκεί. Επίστρεψε τις θέσεις που μένουν μετά την κράτηση ενηλίκων και παιδιών.

```ruby
remaining_seats("20", "6", "3")
# => 11
```

```ruby
remaining_seats("9", "6", "3")
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 002
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 002 --solution
```

Μια μικρή παραλλαγή

Μοίρασε το έγκυρο κείμενο ποσού "7" σε δύο ίσα μέρη με δεκαδική διαίρεση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `3.5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

003 / ΜΑΘΗΜΑ

Κείμενο με δεδομένα · Γέφυρα

Έννοια: Strings, βασικά escapes και interpolation με `#{...}`

Πριν αρχίσεις: 001, 002

Τα διπλά εισαγωγικά επιτρέπουν interpolation: το `#{...}` υπολογίζεται και μπαίνει στο κείμενο. Το `\n` είναι αλλαγή γραμμής, το `\t` tab και το `\"` εισαγωγικό μέσα στο String. Τα μονά εισαγωγικά δεν εκτελούν interpolation.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
name = "Mina"
"Hello #{name}!"
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `confirmation(name, seats, price)`. Επίστρεψε ακριβώς `Όνομα: N | Θέσεις: S | Σύνολο: T`, όπου T είναι seats * price. Τα seats και price είναι μη αρνητικοί ακέραιοι. Μην προσθέσεις αλλαγή γραμμής και μην τυπώσεις.

```ruby
confirmation("Μίνα", 3, 8)
# => "Όνομα: Μίνα | Θέσεις: 3 | Σύνολο: 24"
```

```ruby
confirmation("Άρης", 0, 8)
# => "Όνομα: Άρης | Θέσεις: 0 | Σύνολο: 0"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 003
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 003 --solution
```

Μια μικρή παραλλαγή

Φτιάξε δύο γραμμές: στην πρώτη το όνομα Eva και στη δεύτερη τις 2 θέσεις.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva\n2"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

004 / ΜΑΘΗΜΑ

Μέθοδος που επιστρέφει αποτέλεσμα · Γέφυρα

Έννοια: `def`, παράμετροι, κλήση και τελευταία έκφραση

Πριν αρχίσεις: 002, 003

Το `def label(name)` ορίζει μέθοδο με μία παράμετρο. Ο ορισμός δεν εκτελεί το σώμα: χρειάζεται κλήση, π.χ. `label("Eva")`. Κάθε κλήση έχει τις δικές της παραμέτρους. Χωρίς ρητό `return`, επιστρέφεται η τελευταία έκφραση. Η έξοδος μιας μεθόδου μπορεί να γίνει είσοδος άλλης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def badge(name)
  "[#{name}]"
end
badge("Jo")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `order_total(quantity, unit_price)` για το γινόμενο και `order_label(code, quantity, unit_price)` για κείμενο `CODE: TOTAL`. Η δεύτερη πρέπει να καλεί την πρώτη. Αριθμοί μη αρνητικοί ακέραιοι.

```ruby
order_total(3, 7)
# => 21
```

```ruby
order_label("A2", 3, 7)
# => "A2: 21"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 004
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 004 --solution
```

Μια μικρή παραλλαγή

Όρισε μέθοδο που προσθέτει επίθημα ! σε ένα κείμενο και κάλεσέ την για "Έτοιμο".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Έτοιμο!"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

005 / ΜΑΘΗΜΑ

Επιλογή αποτελέσματος · Γέφυρα

Έννοια: Συγκρίσεις και `if` / `elsif` / `else` ως έκφραση

Πριν αρχίσεις: 004

Το `if` επιλέγει κλάδο και έχει δική του τιμή. Οι έλεγχοι γίνονται από πάνω προς τα κάτω και σταματούν στον πρώτο αληθή. Οι `==`, `<`, `<=`, `>`, `>=` συγκρίνουν τιμές· το `=` παραμένει ανάθεση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
seats = 4
if seats >= 5
  "group"
else
  "small"
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `order_status(paid, quantity)`. Αν quantity == 0 επίστρεψε "κενή". Αλλιώς αν paid είναι true επίστρεψε "έτοιμη". Διαφορετικά "αναμονή". Το paid είναι boolean και quantity μη αρνητικός ακέραιος.

```ruby
order_status(true, 0)
# => "κενή"
```

```ruby
order_status(true, 2)
# => "έτοιμη"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 005
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 005 --solution
```

Μια μικρή παραλλαγή

Δώσε την ετικέτα "μεγάλη" από 10 θέσεις και πάνω, αλλιώς "μικρή". Δοκίμασε ακριβώς το 10.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"μεγάλη"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

006 / ΜΑΘΗΜΑ

Τι θεωρείται αληθές

Έννοια: `nil`, `false`, truthiness και `nil?`

Πριν αρχίσεις: 005

Μόνο `nil` και `false` θεωρούνται ψευδή σε συνθήκη. Το `0` και το `""` είναι truthy. Η `nil?` απαντά αν λείπει τιμή, ανεξάρτητα από το αν η τιμή είναι ψευδής. Αυτό χρειάζεται όταν η επιλογή false είναι κανονικό δεδομένο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
value = 0
if value
  "present"
else
  "absent"
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `choice_label(value)`. Για nil επίστρεψε "λείπει". Για false επίστρεψε "όχι". Για οποιαδήποτε άλλη τιμή επίστρεψε "δόθηκε". Η είσοδος μπορεί να είναι ο ή κενό String.

RUBY
```ruby
choice_label(nil)
# => "λείπει"
```

RUBY
```ruby
choice_label(false)
# => "όχι"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 006
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 006 --solution
```

Μια μικρή παραλλαγή

Επίστρεψε "default" μόνο όταν το value είναι nil, διατηρώντας το false.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `false`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

007 / ΜΑΘΗΜΑ

Σύνδεση κανόνων

Έννοια: `&&`, `||`, `!` και short-circuit evaluation

Πριν αρχίσεις: 006

Το ? στο όνομα entry_allowed? είναι συμβατική ένδειξη ερώτησης, όχι ξεχωριστός τελεστής. Το `&&` σταματά στον πρώτο falsy τελεστέο. Το `||` σταματά στον πρώτο truthy. Επιστρέφουν τελεστέο, που μπορεί να είναι και String ή αριθμός. Το `!` δίνει boolean άρνηση. Με short circuit μπορείς να ελέγξεις ότι υπάρχει τιμή πριν καλέσεις μέθοδό της.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
label = nil
label || "guest"
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `entry_allowed?(paid, banned)` που δίνει true μόνο αν paid και όχι banned. Γράψε και `display_choice(value)` που δίνει value όταν είναι truthy, αλλιώς "default". Τα paid και banned είναι booleans.

```ruby
entry_allowed?(true, false)
# => true
```

```ruby
entry_allowed?(true, true)
# => false
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 007
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 007 --solution
```

Μια μικρή παραλλαγή

Χωρίς if, κράτησε ασφαλή την κλήση nil?: αν υπάρχει κείμενο δώσε την class του, αλλιώς nil.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `nil`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

008 / ΜΑΘΗΜΑ

Πολλές ονομασμένες περιπτώσεις

Έννοια: `case` / `when`

Πριν αρχίσεις: 005

Το `case value` συγκρίνει την τιμή με κάθε `when`. Πολλά στοιχεία χωρισμένα με κόμμα μοιράζονται κλάδο. Το `else` καλύπτει την απουσία αντιστοίχισης. Ο επιλεγμένος κλάδος δίνει την τιμή όλου του case.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
code = "B"
case code
when "A", "B"
  "open"
else
  "closed"
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `status_message(code)`: "new" ή "queued" → "αναμονή", "done" → "έτοιμο", "cancelled" → "ακυρώθηκε", οτιδήποτε άλλο, μαζί με nil → "άγνωστο".

```ruby
status_message("queued")
# => "αναμονή"
```

```ruby
status_message("done")
# => "έτοιμο"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 008
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 008 --solution
```

Μια μικρή παραλλαγή

Ομαδοποίησε κωδικούς 1 και 2 ως "small", με fallback "other". Δοκίμασε το 2.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"small"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

009 / ΜΑΘΗΜΑ

Έξοδος νωρίς

Έννοια: Guard clauses, `return`, modifier `if` και `unless`

Πριν αρχίσεις: 005, 006, 007

Το `return` τελειώνει τη μέθοδο αμέσως. Ένας guard στην αρχή απορρίπτει ειδική περίπτωση και αφήνει τον κανονικό υπολογισμό χωρίς εμφώλευση. Το `return x if condition` είναι modifier if. Το `unless` εκτελεί όταν η συνθήκη είναι falsy.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def seat_fee(n)
  return 0 unless n > 0
  n * 4
end
seat_fee(0)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `booking_cost(seats, price, active)`. Αν active είναι false ή seats <= 0, επίστρεψε 0. Αλλιώς seats * price. Τα seats και price είναι ακέραιοι, price μη αρνητικό και active boolean. Χρησιμοποίησε guards.

RUBY
```ruby
booking_cost(3, 5, false)
# => 0
```

RUBY
```ruby
booking_cost(-1, 5, true)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 009
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 009 --solution
```

Μια μικρή παραλλαγή

Δώσε "guest" μόνο για nil όνομα με guard. Για κενό String κράτησε την τιμή.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `""`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

010 / ΜΑΘΗΜΑ

Κανόνες αποστολής · Εμπέδωση

Έννοια: Εμπέδωση της αρχικής ενότητας

Πριν αρχίσεις: 005, 007, 008, 009

Διάβασε πρώτα ποιοι κανόνες υπερισχύουν. Η τιμή επιστροφής είναι ένα ποσό, άρα η μέθοδος πρέπει να τελειώνει σε αριθμό σε κάθε διαδρομή. Εδώ συνδυάζεις γνωστές συνθήκες και guards.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def pickup_fee(pickup)
  return 0 if pickup
  4
end
pickup_fee(true)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `shipping_fee(weight, region, pickup)`. Πρώτα: αν pickup είναι true δώσε 0. Αλλιώς, για weight <= 0 δώσε nil. Για region "local" βάση 3, για "remote" βάση 7, άλλη περιοχή nil. Αν weight > 5 πρόσθεσε 2 στη βάση. Το weight είναι ακέραιος και pickup boolean.

```ruby
shipping_fee(0, "unknown", true)
# => 0
```

```ruby
shipping_fee(5, "local", false)
# => 3
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 010
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 010 --solution
```

Μια μικρή παραλλαγή

Σε χωριστή μέθοδο δώσε έκπτωση 2 σε ποσό τουλάχιστον 10 και κράτησε τα μικρότερα ποσά ίδια.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `8`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

011 / ΜΑΘΗΜΑ

Διατεταγμένες συλλογές

Έννοια: Array literals, δείκτες, αρνητικοί δείκτες, `length`, `first`, `last`, `empty?`

Πριν αρχίσεις: 005, 006

Ένα Array κρατά τιμές σε σειρά. Το πρώτο στοιχείο έχει δείκτη 0 και το -1 δείχνει το τελευταίο. Δείκτης εκτός ορίων επιστρέφει nil. Οι `first`, `last`, `length` και `empty?` σε βοηθούν να περιγράψεις τη συλλογή χωρίς επανάληψη.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
stops = ["A", "B", "C"]
[stops[0], stops[-1], stops.length]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `route_summary(stops)`. Για κενό Array επίστρεψε "χωρίς στάσεις". Αλλιώς κείμενο `FIRST → LAST (COUNT)`. Τα στοιχεία είναι Strings. Μην αλλάξεις το Array.

```ruby
route_summary(["A", "B", "C"])
# => "A → C (3)"
```

```ruby
route_summary(["A"])
# => "A → A (1)"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 011
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 011 --solution
```

Μια μικρή παραλλαγή

Από τρεις στάσεις δώσε δεύτερη, τελευταία και ένα στοιχείο πέρα από το τέλος.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Y", "Z", nil]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

012 / ΜΑΘΗΜΑ

Συγκέντρωση τιμών

Έννοια: Νέος πίνακας, `<<`, `push` και συνένωση με `+`

Πριν αρχίσεις: 011

Το `[]` δημιουργεί νέο Array. Τα `<<` και `push` προσθέτουν στο ίδιο Array και το επιστρέφουν. Το `+` ενώνει δύο Arrays σε νέο εξωτερικό Array. Εδώ χτίζουμε δικό μας αποτέλεσμα, οπότε μπορούμε να το μεταβάλλουμε.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
options = ["tea"]
options << "water"
options + ["juice"]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `menu_options(coffee, dessert)`. Πάντα πρώτη επιλογή "νερό", μετά "καφές" αν coffee είναι true και μετά "γλυκό" αν dessert είναι true. Επίστρεψε νέο Array σε κάθε κλήση. Οι είσοδοι είναι booleans.

```ruby
menu_options(true, true)
# => ["νερό", "καφές", "γλυκό"]
```

```ruby
menu_options(false, true)
# => ["νερό", "γλυκό"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 012
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 012 --solution
```

Μια μικρή παραλλαγή

Ένωσε αρχικές και έξτρα επιλογές και έλεγξε ότι το αρχικό Array παραμένει ίδιο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[["tea"], ["tea", "cake"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

013 / ΜΑΘΗΜΑ

Ένα block για κάθε στοιχείο

Έννοια: `each`, `do ... end`, `{ ... }`, παράμετρος block

Πριν αρχίσεις: 011, 012

Το block `do |item| ... end` είναι κώδικας που δίνεις στη μέθοδο. Η `each` το εκτελεί μία φορά ανά στοιχείο, με την τρέχουσα τιμή στην παράμετρο item. Για σύντομο block γράφουμε `{ |item| ... }`. Η Array#each επιστρέφει το αρχικό Array, όχι τις επιστροφές των blocks.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
total = 0
[2, 4, 1].each do |seats|
  total = total + seats
end
total
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `total_seats(bookings)` για Array μη αρνητικών ακεραίων. Χρησιμοποίησε each και αρχικό total = 0. Επίστρεψε άθροισμα θέσεων, χωρίς αλλαγή της εισόδου.

RUBY
```ruby
total_seats([2, 4, 2])
# => 8
```

RUBY
```ruby
total_seats([])
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 013
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 013 --solution
```

Μια μικρή παραλλαγή

Με each χτίσε νέο Array με το κείμενο "θέσεις=N" για κάθε κράτηση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["θέσεις=1", "θέσεις=3"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

014 / ΜΑΘΗΜΑ

Μία νέα τιμή για κάθε παλιά

Νέα έννοια: `map` και η τιμή που επιστρέφει το block.

Πριν αρχίσεις: χρειάζονται μέθοδοι και επιστροφή αποτελέσματος, interpolation, `if` / `elsif`, Arrays και το block της `each`. Στον χάρτη έχουν προηγηθεί τα 001-013.

Η `map` είναι χρήσιμη όταν κάθε στοιχείο της εισόδου πρέπει να δώσει ένα στοιχείο στην έξοδο. Ας φτιάξουμε μια ετικέτα για κάθε αριθμό δωματίου.

RUBY
```ruby
rooms = [12, 4, 18]

labels = rooms.map do |room|
  "Δωμάτιο #{room}"
end

p labels
# => ["Δωμάτιο 12", "Δωμάτιο 4", "Δωμάτιο 18"]
```

Η `room` παίρνει διαδοχικά τις τιμές `12`, `4`, `18`. Η τελευταία έκφραση του block είναι το κείμενο της ετικέτας. Η `map` βάζει αυτές τις τρεις τιμές σε νέο Array και τον επιστρέφει.

Η έξοδος έχει ίδιο πλήθος και αντίστοιχη σειρά στοιχείων με την είσοδο. Για κενή είσοδο, είναι κενή. Αυτή είναι η συμπεριφορά του Array#map.

Το block μπορεί να περιέχει περισσότερες γραμμές και συνθήκες. Η τιμή που καταλήγει στο νέο Array είναι η τελική του τιμή σε κάθε εκτέλεση. Η προηγούμενη `each` εκτελεί επίσης block για κάθε στοιχείο, αλλά δεν συγκεντρώνει τις επιστροφές του σε νέο Array.

Πριν συνεχίσεις, πρόβλεψε το αποτέλεσμα.

RUBY
```ruby
numbers = [2, 0, 5]

result = numbers.map do |number|
  number * 3
  number + 1
end
```

Η απάντηση είναι `[3, 1, 6]`. Η τελευταία έκφραση είναι το `number + 1`. Ο προηγούμενος υπολογισμός δεν είναι η επιστροφή του block.

Η άσκηση

Γράψε τη `grade_labels(scores)`. Δέχεται Array με ακέραιους βαθμούς από 0 έως 100 και επιστρέφει νέο Array με μία ετικέτα για κάθε βαθμό.

- Από 90 έως 100: `"άριστα"`.
- Από 50 έως 89: `"πέρασε"`.

- Από 0 έως 49: `"κόπηκε"`.

Χρησιμοποίησε `map`. Κράτησε τη σειρά και τις επαναλήψεις. Μην αλλάξεις το `scores` και μην εμφανίσεις την απάντηση με `puts` ή `p`. Η μέθοδος πρέπει να την επιστρέψει.

Η είσοδος είναι έγκυρη. Δεν χρειάζεται έλεγχος τύπων ή μήνυμα σφάλματος.

Κλήση	Αποτέλεσμα
`grade_labels([49, 50, 89, 90])`	`["κόπηκε", "πέρασε", "πέρασε", "άριστα"]`
`grade_labels([100, 0, 100])`	`["άριστα", "κόπηκε", "άριστα"]`
`grade_labels([])`	`[]`

Γράψε στο exercise.rb. Τα tests είναι ήδη έτοιμα. Από τον φάκελο `syntax-first/ruby`.

```sh
bin/lesson 014
```

Η αρχική προσπάθεια έχει `TODO` και αποτυγχάνει μέχρι να τη συμπληρώσεις.

Μια μικρή παραλλαγή

Σε δεύτερη μέθοδο `detailed_grade_labels(scores)`, κράτησε τους ίδιους κανόνες και πρόσθεσε τον βαθμό στην ετικέτα.

```ruby
detailed_grade_labels([49, 90])
# => ["49: κόπηκε", "90: άριστα"]
```

Χρειάζεσαι μόνο το interpolation που ήδη γνωρίζεις. Η παραλλαγή είναι χωριστή από την κύρια προσπάθεια και τα tests της.

Αν κολλήσεις

Πρώτα γράψε μέσα στο block το `if` που θα επέστρεφε τη σωστή ετικέτα για έναν βαθμό. Έπειτα έλεγξε ποια είναι η τελευταία έκφραση του block. Η `map` θα αναλάβει να μαζέψει τις απαντήσεις.

015 / ΜΑΘΗΜΑ

Κράτησε όσα ταιριάζουν

Έννοια: `select`

Πριν αρχίσεις: 007, 013, 014

Η `select` επιστρέφει νέο Array με τα αρχικά στοιχεία για τα οποία το block δίνει truthy τιμή. Η παράμετρος του block είναι ένα στοιχείο. Δεν μετατρέπει τις τιμές και κρατά σειρά και επαναλήψεις.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[2, 8, 4, 8].select { |n| n >= 5 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `available_quantities(quantities, minimum, maximum)` με select. Οι ποσότητες και τα όρια είναι ακέραιοι, minimum <= maximum. Κράτησε όσα βρίσκονται μέσα στο κλειστό διάστημα. Η είσοδος μένει ίδια και το εξωτερικό αποτέλεσμα είναι νέο Array.

```ruby
available_quantities([1, 2, 4, 5, 4], 2, 4)
# => [2, 4, 4]
```

```ruby
available_quantities([1, 9], 2, 4)
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 015
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 015 --solution
```

Μια μικρή παραλλαγή

Κράτησε ποσότητες που ισούνται ακριβώς με τον στόχο 4.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[4, 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

016 / ΜΑΘΗΜΑ

Αφαίρεσε όσα απορρίπτονται

Νέα έννοια: `reject`.

Πριν αρχίσεις: χρειάζονται Array, block, συγκρίσεις, `||` και `select`. Στον χάρτη έχουν προηγηθεί τα 001-015.

Μια λίστα μπορεί να περιέχει τιμές που θέλεις να αφαιρέσεις. Με τη `reject` γράφεις τη συνθήκη απόρριψης.

```ruby
quantities = [4, 0, 7, 0, 2]

available = quantities.reject do |quantity|
  quantity == 0
end

p available
# => [4, 7, 2]
```

Το block απαντά για κάθε ποσότητα αν πρέπει να απορριφθεί. Η ποσότητα `0` κάνει τη συνθήκη αληθή, οπότε αφαιρείται. Οι υπόλοιπες ποσότητες παραμένουν οι αρχικές τους τιμές.

Η `reject` επιστρέφει νέο Array και κρατά τη σχετική σειρά όσων παραμένουν. Η συνθήκη του block δεν γίνεται στοιχείο της εξόδου. Αυτή είναι η συμπεριφορά του Array#reject.

Σύγκρινε τα δύο blocks.

```ruby
numbers = [2, 8, 3]

kept = numbers.select do |number|
  number > 5
end
# => [8]

remaining = numbers.reject do |number|
  number > 5
end
# => [2, 3]
```

Η `select` κρατά όταν η συνθήκη είναι αληθής. Η `reject` απορρίπτει όταν είναι αληθής. Πρόβλεψε τώρα το αποτέλεσμα για είσοδο `[9, 9, 1]` στο δεύτερο block: θα παραμείνει το `[1]`.

Η άσκηση

Γράψε τη `valid_scores(scores, minimum, maximum)`. Αφαίρεσε κάθε τιμή μικρότερη από το `minimum` ή μεγαλύτερη από το `maximum`.

Τα ίδια τα όρια είναι αποδεκτά. Χρησιμοποίησε `reject`. Επίστρεψε νέο Array με την αρχική σειρά και τα αρχικά διπλότυπα. Μην αλλάξεις το `scores`.

Όλες οι τιμές και τα όρια είναι ακέραιοι. Το `minimum` είναι μικρότερο ή ίσο από το `maximum`. Επιτρέπονται αρνητικές τιμές και κενή είσοδος. Δεν χρειάζεται έλεγχος τύπων ή ορίων των παραμέτρων.

Κλήση	Αποτέλεσμα
`valid_scores([-1, 0, 30, 100, 101], 0, 100)`	`[0, 30, 100]`
`valid_scores([20, 10, 20, 9], 10, 20)`	`[20, 10, 20]`
`valid_scores([4, 7, 4], 4, 4)`	`[4, 4]`
`valid_scores([], 0, 100)`	`[]`

Γράψε στο exercise.rb. Από τον φάκελο `syntax-first/ruby`.

```sh
bin/lesson 016
```

Μια μικρή παραλλαγή

Γράψε δεύτερη μέθοδο `invalid_scores(scores, minimum, maximum)` που χρησιμοποιεί πάλι `reject`, αλλά αυτή τη φορά επιστρέφει μόνο τις άκυρες τιμές.

```ruby
invalid_scores([-1, 0, 30, 100, 101], 0, 100)
# => [-1, 101]
```

Η αλλαγή είναι στο τι περιγράφει η συνθήκη απόρριψης. Η παραλλαγή είναι χωριστή από την κύρια προσπάθεια και τα tests της.

Αν κολλήσεις

Πες με λόγια πότε ένας βαθμός είναι άκυρος. Μετέτρεψε αυτή την πρόταση σε συνθήκη με τα εργαλεία που γνωρίζεις ήδη. Πρόσεξε αν τα όρια πρέπει να παραμείνουν.

017 / ΜΑΘΗΜΑ

Μετατροπή ή φίλτρο; · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 014, 015, 016

Πριν επιλέξεις μέθοδο, όρισε το σχήμα εξόδου. Θέλεις μία τιμή ανά είσοδο ή μόνο μερικές από τις αρχικές τιμές; Η άσκηση ζητά τρεις ανεξάρτητες συλλογές ώστε να φαίνεται η διαφορά.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
scores = [40, 80]
[scores.map { |n| n + 1 }, scores.select { |n| n >= 50 }]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `split_scores(scores)`. Για Array ακεραίων δώσε Array τριών Arrays: ετικέτες "εντός"/"εκτός" για κάθε βαθμό, βαθμούς μέσα στο 0..100 και βαθμούς έξω από αυτό. Κράτησε σειρά και διπλότυπα, χωρίς αλλαγή εισόδου. Γράψε τους ελέγχους με συγκρίσεις.

```ruby
split_scores([-1, 50, 101, 50])
# => [["εκτός", "εντός", "εκτός", "εντός"], [50, 50], [-1, 101]]
```

```ruby
split_scores([0, 100])
# => [["εντός", "εντός"], [0, 100], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 017
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 017 --solution
```

Μια μικρή παραλλαγή

Κράτησε πρώτα βαθμούς τουλάχιστον 50 και μετά πρόσθεσε 5 στον καθένα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[55, 85]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

018 / ΜΑΘΗΜΑ

Το πρώτο που ταιριάζει

Έννοια: `find` και αποτέλεσμα `nil`

Πριν αρχίσεις: 006, 013, 015

Η `find` σταματά στην πρώτη truthy απάντηση του block και επιστρέφει εκείνο το αρχικό στοιχείο. Αν δεν βρει τίποτα, δίνει nil. Δεν επιστρέφει Array. Έλεγξε nil? όταν το 0 μπορεί να είναι έγκυρο αποτέλεσμα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[3, 8, 6].find { |n| n >= 5 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `first_stock(stocks, required)` που επιστρέφει την πρώτη ακέραιη ποσότητα >= required ή nil. Μην ταξινομήσεις και μην αλλάξεις τη λίστα. Όλες οι ποσότητες και required είναι μη αρνητικά.

```ruby
first_stock([2, 9, 6], 5)
# => 9
```

```ruby
first_stock([1, 2], 3)
# => nil
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 018
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 018 --solution
```

Μια μικρή παραλλαγή

Μετέτρεψε απουσία αποτελέσματος σε "δεν βρέθηκε", κρατώντας την ποσότητα όταν υπάρχει.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"δεν βρέθηκε"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

019 / ΜΑΘΗΜΑ

Ερωτήσεις για όλη τη συλλογή

Έννοια: `any?`, `all?`, `none?`, `one?`

Πριν αρχίσεις: 006, 013

Οι `any?`, `all?`, `none?`, `one?` απαντούν αντίστοιχα: τουλάχιστον ένα, όλα, κανένα, ακριβώς ένα. Κάθε block παίρνει ένα στοιχείο και ελέγχει τον κανόνα. Στην κενή συλλογή οι all? και none? είναι true, οι άλλες false.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = [0, 3]
[values.any? { |n| n > 0 }, values.all? { |n| n > 0 }]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `passing_questions(scores)` για ακέραιους βαθμούς. Επιτυχία σημαίνει >= 50. Επίστρεψε τέσσερα booleans με σειρά any, all, none, one για τον ίδιο κανόνα.

```ruby
passing_questions([40, 50])
# => [true, false, false, true]
```

```ruby
passing_questions([50, 80])
# => [true, true, false, false]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 019
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 019 --solution
```

Μια μικρή παραλλαγή

Έλεγξε αν υπάρχει ακριβώς μία κενή λίστα μέσα σε δύο λίστες.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

020 / ΜΑΘΗΜΑ

Πλήθος με κριτήριο

Έννοια: `count` με block

Πριν αρχίσεις: 013, 015

Η `count` με block επιστρέφει πόσες φορές ο κανόνας έδωσε truthy αποτέλεσμα. Έτσι δεν χρειάζεται να φτιάξεις λίστα μόνο για να μετρήσεις το μήκος της. Η απάντηση είναι ακέραιος, με 0 για καμία αντιστοίχιση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[2, 5, 2].count { |n| n == 2 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `fitting_bookings(bookings, capacity)`. Μέτρησε κρατήσεις με θετικές θέσεις και θέσεις <= capacity. Οι είσοδοι είναι μη αρνητικοί ακέραιοι. Κάθε επανάληψη μετρά ως χωριστή κράτηση.

```ruby
fitting_bookings([0, 2, 4, 4, 5], 4)
# => 3
```

```ruby
fitting_bookings([0, 1], 0)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 020
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 020 --solution
```

Μια μικρή παραλλαγή

Μέτρησε τις κρατήσεις που υπερβαίνουν χωρητικότητα 3.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `2`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

021 / ΜΑΘΗΜΑ

Άθροισμα με μετατροπή

Έννοια: `sum` και block

Πριν αρχίσεις: 013, 014

Η `sum` προσθέτει στοιχεία. Με block προσθέτει τις τιμές που υπολογίζει το block. Η αρχική τιμή είναι 0, αλλά μπορείς να δώσεις άλλη με `sum(initial)`. Στην κενή είσοδο επιστρέφεται η αρχική τιμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[2, 3].sum { |n| n * 4 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `booking_revenue(bookings, price, fee)`. Κάθε κράτηση κοστίζει seats * price + fee, ακόμη και αν seats είναι 0. Επίστρεψε το άθροισμα με sum block. Όλοι οι αριθμοί μη αρνητικοί ακέραιοι.

```ruby
booking_revenue([2, 3], 5, 1)
# => 27
```

```ruby
booking_revenue([0], 5, 1)
# => 1
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 021
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 021 --solution
```

Μια μικρή παραλλαγή

Άρχισε από πάγιο 10 και πρόσθεσε τις δοσμένες χρεώσεις 2 και 3.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `15`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

022 / ΜΑΘΗΜΑ

Θέση μαζί με τιμή

Έννοια: `each_with_index`

Πριν αρχίσεις: 003, 012, 013

Η `each_with_index` δίνει δύο παραμέτρους: το στοιχείο και τον δείκτη από 0. Η θέση που εμφανίζουμε στον άνθρωπο συχνά είναι index + 1. Χτίζουμε νέο Array με <<, όπως στο 013.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
rows = []
["A", "B"].each_with_index { |name, index| rows << "#{index + 1}. #{name}" }
rows
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `ranking_lines(names)`. Επίστρεψε Array Strings της μορφής `1. NAME`, `2. NAME` κλπ. Διατήρησε σειρά και επαναλήψεις χωρίς να αλλάξεις names.

```ruby
ranking_lines(["Eva", "Jo", "Eva"])
# => ["1. Eva", "2. Jo", "3. Eva"]
```

```ruby
ranking_lines([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 022
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 022 --solution
```

Μια μικρή παραλλαγή

Φτιάξε αρίθμηση από το 10 με την ίδια each_with_index.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["10: A", "11: B"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

023 / ΜΑΘΗΜΑ

Μικρό βαθμολόγιο · Εμπέδωση

Έννοια: Επιλογή εργαλείων από τα προηγούμενα μαθήματα

Πριν αρχίσεις: 014, 015, 020

Μια μικρή αναφορά χρειάζεται πρώτα καθαρά δεδομένα και μετά αποτελέσματα πάνω σε αυτά. Διάλεξε ξεχωριστά το φίλτρο, τη μετατροπή και τη μέτρηση. Δεν χρειάζεται νέος κανόνας επανάληψης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
valid = [-1, 40, 80].select { |n| n >= 0 }
[valid, valid.count { |n| n >= 50 }]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `gradebook(scores)` για ακέραιους. Κράτησε έγκυρους 0 έως 100. Επίστρεψε [valid, labels, passed_count], όπου labels είναι "πέρασε" για >=50, αλλιώς "κόπηκε", μόνο για τους έγκυρους. Η είσοδος δεν αλλάζει.

```ruby
gradebook([-1, 49, 50, 101, 100])
# => [[49, 50, 100], ["κόπηκε", "πέρασε", "πέρασε"], 2]
```

```ruby
gradebook([-1, 101])
# => [[], [], 0]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 023
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 023 --solution
```

Μια μικρή παραλλαγή

Στους έγκυρους βαθμούς [0, 50, 100] μέτρησε πόσοι είναι κάτω από 50.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `1`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

024 / ΜΑΘΗΜΑ

Καθάρισμα άκρων

Έννοια: `strip`, `lstrip`, `rstrip`, `chomp` και `empty?`

Πριν αρχίσεις: 023

Η `strip` αφαιρεί whitespace και από τα δύο άκρα, οι `lstrip` και `rstrip` από ένα. Η `chomp` αφαιρεί έναν τελικό διαχωριστή γραμμής. Οι εκδοχές χωρίς ! επιστρέφουν νέο String. Τα εσωτερικά κενά μένουν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
"  A  B\n".strip
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `clean_names(names)`. Για Array Strings επίστρεψε τις καθαρισμένες τιμές με strip. Κράτησε και τα κενά αποτελέσματα, τη σειρά και την είσοδο ίδια.

```ruby
clean_names([" A  B ", "\tJo\n"])
# => ["A  B", "Jo"]
```

```ruby
clean_names(["  ", ""])
# => ["", ""]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 024
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 024 --solution
```

Μια μικρή παραλλαγή

Αφαίρεσε μόνο το newline από " A \n".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `" A "`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

025 / ΜΑΘΗΜΑ

Πεζά και κεφαλαία

Έννοια: `downcase`, `upcase`, `capitalize`

Πριν αρχίσεις: 014, 024

Οι `downcase` και `upcase` αλλάζουν πεζά/κεφαλαία. Η `capitalize` κάνει κεφαλαίο τον πρώτο χαρακτήρα και πεζούς τους επόμενους. Επιστρέφουν νέα Strings. Δεν κεφαλαιοποιεί κάθε λέξη η capitalize.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
"rUBY CODE".capitalize
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `code_labels(codes)`. Για Array ASCII κωδικών Strings, καθάρισε τα άκρα και επίστρεψε κεφαλαίες τιμές, χωρίς αλλαγή της εισόδου.

```ruby
code_labels([" ab ", "cD"])
# => ["AB", "CD"]
```

```ruby
code_labels(["", "ab", "ab"])
# => ["", "AB", "AB"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 025
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 025 --solution
```

Μια μικρή παραλλαγή

Φτιάξε πεζούς κωδικούς από [" AB ", "Cd"].

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["ab", "cd"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

026 / ΜΑΘΗΜΑ

Κείμενο σε πεδία

Έννοια: `split` και επιλογή διαχωριστή

Πριν αρχίσεις: 011, 014, 024

Η `split(separator)` χωρίζει String σε Array. Χωρίς όρισμα χωρίζει σε whitespace. Με ρητό ":" χωρίζει εκεί. Το προαιρετικό δεύτερο όρισμα περιορίζει τα πεδία· αρνητικό όριο κρατά και τελικά κενά πεδία.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
"A:2".split(":")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `parse_quantities(lines)`. Κάθε γραμμή έχει ακριβώς τη μορφή NAME:INTEGER, μη κενό όνομα χωρίς : και έγκυρο μη αρνητικό αριθμό. Επίστρεψε Arrays [όνομα, ακέραιη ποσότητα].

```ruby
parse_quantities(["Eva:02", "Jo:0"])
# => [["Eva", 2], ["Jo", 0]]
```

```ruby
parse_quantities([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 026
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 026 --solution
```

Μια μικρή παραλλαγή

Χώρισε το "A:" κρατώντας το τελικό κενό πεδίο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["A", ""]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

027 / ΜΑΘΗΜΑ

Πεδία σε κείμενο

Έννοια: `join`

Πριν αρχίσεις: 014, 026

Η `join(separator)` ενώνει τις τιμές Array σε String. Ο διαχωριστής μπαίνει ανάμεσα στα στοιχεία, όχι στο τέλος. Κενό Array δίνει κενό String. Έτσι ορίζεις ξεχωριστά το περιεχόμενο γραμμής και το πώς ενώνονται οι γραμμές.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["tea", "cake"].join(" / ")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `text_report(rows)`. Κάθε row είναι Array Strings. Ένωσε τα πεδία με " | " και τις γραμμές με newline. Χωρίς τελικό newline. Η είσοδος παραμένει ίδια.

```ruby
text_report([["A", "2"], ["B", "0"]])
# => "A | 2\nB | 0"
```

```ruby
text_report([])
# => ""
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 027
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 027 --solution
```

Μια μικρή παραλλαγή

Ένωσε ονόματα με κόμμα και κενό.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva, Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

028 / ΜΑΘΗΜΑ

Διαστήματα τιμών

Έννοια: `..`, `...`, `cover?` και `to_a`

Πριν αρχίσεις: 005, 011

Το `a..b` περιλαμβάνει το b, ενώ `a...b` το αποκλείει. Η `cover?` ελέγχει τα όρια και η `to_a` υλοποιεί ένα πεπερασμένο διακριτό Range σε Array. Χρησιμοποίησε παρενθέσεις πριν καλέσεις μέθοδο σε Range.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
[(2..4).to_a, (2...4).to_a]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `discount_band(quantity)`. Για μη αρνητικό ακέραιο quantity δώσε "small" στο 0..4, "medium" στο 5..9 και "large" από 10. Χρησιμοποίησε cover?.

RUBY
```ruby
discount_band(0)
# => "small"
```

RUBY
```ruby
discount_band(5)
# => "medium"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 028
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 028 --solution
```

Μια μικρή παραλλαγή

Παρήγαγε τις θέσεις 3 έως 6, χωρίς το 6.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[3, 4, 5]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

029 / ΜΑΘΗΜΑ

Κομμάτια από μια τιμή

Έννοια: Slicing σε String και Array με δείκτη, μήκος ή Range

Πριν αρχίσεις: 011, 028

Το `value[start, length]` ζητά πλήθος στοιχείων, ενώ το `value[range]` ζητά διάστημα δεικτών. Ισχύει σε Arrays και Strings. Στην ακριβή θέση τέλους με μήκος 0 παίρνεις κενό· πέρα από το τέλος παίρνεις nil.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[["A", "B", "C"][1, 2], "ABCDE"[1...3]]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `result_page(items, offset, size)`. Με μη αρνητικούς ακέραιους offset και size, επίστρεψε το ζητούμενο κομμάτι Array. Αν offset είναι πέρα από το τέλος, επίστρεψε []. Μην αλλάξεις την είσοδο.

```ruby
result_page([1, 2, 3], 1, 9)
# => [2, 3]
```

```ruby
result_page([1], 3, 2)
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 029
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 029 --solution
```

Μια μικρή παραλλαγή

Πάρε τους τρεις πρώτους χαρακτήρες του "RUBY-42".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"RUB"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

030 / ΜΑΘΗΜΑ

Απλή αναζήτηση κειμένου

Έννοια: `include?`, `start_with?`, `end_with?`

Πριν αρχίσεις: 007, 015

Η `include?` ελέγχει αν υπάρχει ένα υποκείμενο οπουδήποτε. Οι `start_with?` και `end_with?` ελέγχουν τα άκρα. Είναι ακριβείς και case sensitive συγκρίσεις Strings. Δεν χρειάζεται regex για σταθερά προθέματα και καταλήξεις.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["report.csv".end_with?(".csv"), "x-report.csv".start_with?("report")]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `matching_files(names, prefix, suffix)`. Κράτησε τα ονόματα Strings που αρχίζουν με prefix και τελειώνουν σε suffix. Δεν διαβάζεις πραγματικά αρχεία. Διατήρησε σειρά, επαναλήψεις και είσοδο.

```ruby
matching_files(["run-a.csv", "x-run.csv", "run-b.txt"], "run-", ".csv")
# => ["run-a.csv"]
```

```ruby
matching_files(["Run-a.csv"], "run-", ".csv")
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 030
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 030 --solution
```

Μια μικρή παραλλαγή

Κράτησε ονόματα που περιέχουν "draft" οπουδήποτε.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["my-draft.txt"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

031 / ΜΑΘΗΜΑ

Αντικατάσταση σταθερού κειμένου

Έννοια: `sub` και `gsub` με String

Πριν αρχίσεις: 003

Η `sub(old, new)` αλλάζει μόνο την πρώτη αντιστοίχιση. Η `gsub` αλλάζει όλες. Με String pattern αναζητούν σταθερό κείμενο. Οι εκδοχές χωρίς ! επιστρέφουν String χωρίς να αλλάζουν την είσοδο. Στην άσκηση τα replacements δεν περιέχουν backslash.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["X-X".sub("X", "A"), "X-X".gsub("X", "A")]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `fill_template(text, name)`. Αντικατάστησε όλα τα σταθερά tokens {name} με name. Δέχεται Strings, χωρίς backslash στο name. Κράτησε το αρχικό text ίδιο και άφησε άλλα tokens όπως είναι.

```ruby
fill_template("Hi {name}, bye {name}", "Eva")
# => "Hi Eva, bye Eva"
```

```ruby
fill_template("{date}: {name}", "Jo")
# => "{date}: Jo"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 031
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 031 --solution
```

Μια μικρή παραλλαγή

Αντικατάστησε μόνο το πρώτο "TODO" με "DONE".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"DONE, TODO"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

032 / ΜΑΘΗΜΑ

Χαρακτήρες και γραμμές

Έννοια: `chars`, `each_char`, `lines`, `each_line`

Πριν αρχίσεις: 020, 022, 024, 027

Η `chars` δίνει Array χαρακτήρων και η `lines` Array γραμμών. Οι `each_char` και `each_line` με block διασχίζουν χωρίς δικό σου split. Οι γραμμές διατηρούν συνήθως το τελικό newline, άρα χρησιμοποίησε chomp όταν θέλεις μόνο το περιεχόμενο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
"A\nB\n".lines.map { |line| line.chomp }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `numbered_text(text)`. Αρίθμησε κάθε γραμμή String από 1 ως `N: TEXT`. Αφαίρεσε μόνο το τέλος γραμμής με chomp. Επίστρεψε Array Strings. Κενό κείμενο δίνει [].

RUBY
```ruby
numbered_text("A\n\nB")
# => ["1: A", "2: ", "3: B"]
```

RUBY
```ruby
numbered_text("A\n")
# => ["1: A"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 032
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 032 --solution
```

Μια μικρή παραλλαγή

Με each_char μέτρησε πόσες φορές υπάρχει ο χαρακτήρας a στο "aba".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `2`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

033 / ΜΑΘΗΜΑ

Καθάρισμα λίστας ονομάτων · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 014, 016, 024, 025, 027

Η σειρά των μετατροπών έχει σημασία. Πρώτα καθάρισε για να φανεί ποια ονόματα είναι κενά, έπειτα κράτησε τις χρήσιμες τιμές και τέλος παρουσίασέ τες.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[" A ", " "].map { |s| s.strip }.reject { |s| s.empty? }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `name_lines(names)`. Καθάρισε τα άκρα, κάνε capitalize, αφαίρεσε όσα γίνουν κενά και ένωσε με newline. Χωρίς τελικό newline ή αλλαγή εισόδου. Για την άσκηση τα ονόματα είναι ASCII Strings.

```ruby
name_lines([" eVA ", " ", "JO"])
# => "Eva\nJo"
```

```ruby
name_lines([" ", ""])
# => ""
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 033
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 033 --solution
```

Μια μικρή παραλλαγή

Με τα ίδια βήματα φτιάξε μία γραμμή με διαχωριστή ", ".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva, Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

034 / ΜΑΘΗΜΑ

Μεταβολή του ίδιου αντικειμένου

Έννοια: Μεταβαλλόμενες τιμές, bang methods και πιθανό αποτέλεσμα `nil`

Πριν αρχίσεις: 024, 025

Η `strip!` και η `upcase!` αλλάζουν τον receiver. Σε αυτές τις μεθόδους το αποτέλεσμα είναι nil όταν δεν χρειάστηκε αλλαγή. Το String#<< προσθέτει κείμενο στο ίδιο String. Το ! είναι σύμβαση ονόματος, όχι γενικός κανόνας επιστροφής για κάθε μέθοδο της Ruby.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
text = "abc"
returned = text.strip!
[text, returned]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `normalize_in_place(text)`. Καθάρισε άκρα και κάνε κεφαλαίο το ίδιο String με bang methods. Επίστρεφε πάντα το ίδιο text αντικείμενο, ακόμη κι αν ήταν ήδη καθαρό. Εδώ επιτρέπεται ρητά η μεταβολή της εισόδου.

RUBY
```ruby
normalize_in_place(" ab ")
# => "AB"
```

RUBY
```ruby
normalize_in_place("AB")
# => "AB"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 034
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 034 --solution
```

Μια μικρή παραλλαγή

Παρατήρησε τι επιστρέφει η upcase! όταν το κείμενο είναι ήδη κεφαλαίο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[nil, "OK"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

035 / ΜΑΘΗΜΑ

Δύο ονόματα, ένα αντικείμενο

Έννοια: Αναφορές και ρηχό αντίγραφο με `dup`

Πριν αρχίσεις: 012, 034

Η ανάθεση άλλου ονόματος δεν αντιγράφει αντικείμενο. Η `dup` φτιάχνει ρηχό αντίγραφο: νέο εξωτερικό αντικείμενο, κοινά εσωτερικά στοιχεία. Αν αντιγράψεις Array Strings, τα Strings παραμένουν κοινά μέχρι να τα αντιγράψεις ξεχωριστά.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
original = ["A"]
copy = original.dup
copy[0] << "!"
[original, copy]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `loud_title(title)`. Επίστρεψε κεφαλαίο αντίγραφο του String title με dup και upcase!. Μην αλλάξεις το αρχικό. Το αποτέλεσμα πρέπει να είναι άλλο String και όταν είναι ήδη κεφαλαίο.

```ruby
loud_title("Ruby")
# => "RUBY"
```

```ruby
s = "Ruby"
loud_title(s)
s
# => "Ruby"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 035
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 035 --solution
```

Μια μικρή παραλλαγή

Αντίγραψε και κάθε εσωτερικό String ώστε η προσθήκη ! στο αντίγραφο να μη μεταβάλλει το αρχικό Array.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[["A"], ["A!"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

036 / ΜΑΘΗΜΑ

Καθαρές ετικέτες συμμετοχής · Εμπέδωση

Έννοια: Σύνθεση strings, Arrays, `map` και `reject`

Πριν αρχίσεις: 014, 016, 024, 025, 035

Συνδύασε καθάρισμα, φίλτρο και παρουσίαση. Η εκφώνηση ορίζει ότι οι αρχικές τιμές διατηρούνται, άρα διάλεξε μεθόδους χωρίς μεταβολή της εισόδου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
[" jo "].map { |s| "[#{s.strip.upcase}]" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `participant_badges(names)`. Για ASCII ονόματα Strings επίστρεψε νέες ετικέτες `[NAME]` με καθαρό κεφαλαίο όνομα. Αφαίρεσε κενά ονόματα μετά το καθάρισμα. Διατήρησε σειρά και αρχικά Strings.

RUBY
```ruby
participant_badges([" eva ", " ", "jo"])
# => ["[EVA]", "[JO]"]
```

RUBY
```ruby
participant_badges([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 036
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 036 --solution
```

Μια μικρή παραλλαγή

Φτιάξε πεζές ετικέτες με πρόθεμα @, για τα ίδια είδη εισόδου.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["@eva"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

037 / ΜΑΘΗΜΑ

Εγγραφές με ονομασμένα πεδία

Έννοια: Hash literals, Symbols, `%i`, `[]` και ενημέρωση τιμής

Πριν αρχίσεις: 003, 011

Το Hash συνδέει κλειδιά με τιμές. Για String κλειδί γράφεις `{"name" => "Eva"}`. Το `{name: "Eva"}` έχει Symbol κλειδί `:name`, που διαφέρει από το String "name". Η σύνταξη `%i[name seats]` φτιάχνει Array Symbols. Με `record[:seats] = 3` αλλάζεις ένα πεδίο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
record = {name: "Eva", seats: 2}
record[:seats] = 3
[record[:name], record[:seats], %i[name seats]]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `booking_record(name, seats)`. Φτιάξε νέο Hash με Symbol κλειδιά name, seats, status. Το status είναι "pending". Επίστρεψε [record, description], με description `NAME: SEATS (pending)`. Το name είναι String και seats μη αρνητικός ακέραιος.

RUBY
```ruby
booking_record("Eva", 2)
# => [{name: "Eva", seats: 2, status: "pending"}, "Eva: 2 (pending)"]
```

RUBY
```ruby
booking_record("Jo", 0)
# => [{name: "Jo", seats: 0, status: "pending"}, "Jo: 0 (pending)"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 037
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 037 --solution
```

Μια μικρή παραλλαγή

Άλλαξε μόνο την κατάσταση μιας νέας εγγραφής σε "ready".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{name: "Jo", status: "ready"}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

038 / ΜΑΘΗΜΑ

Λείπει κλειδί ή υπάρχει τιμή nil;

Έννοια: `key?` και `fetch` με ρητή προεπιλογή

Πριν αρχίσεις: 006, 037

Η `key?` ελέγχει παρουσία κλειδιού. Η `fetch(key, default)` χρησιμοποιεί default μόνο αν λείπει το κλειδί. Χωρίς default, η fetch προκαλεί KeyError για απόν κλειδί. Το `[]` επιστρέφει nil και για απουσία και για αποθηκευμένο nil, άρα μόνο του δεν διακρίνει αυτές τις περιπτώσεις.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
settings = {enabled: false, title: nil}
[settings.fetch(:enabled, true), settings.fetch(:title, "X"), settings.fetch(:size, 4)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `effective_settings(settings)`. Επίστρεψε νέο Hash με enabled default true και title default "Guest". Κράτησε ρητές τιμές false ή nil. Πρόσθεσε title_given που δείχνει αν υπήρχε :title. Η είσοδος είναι Hash Symbol κλειδιών και μένει ίδια.

RUBY
```ruby
effective_settings({})
# => {enabled: true, title: "Guest", title_given: false}
```

RUBY
```ruby
effective_settings({enabled: false, title: nil})
# => {enabled: false, title: nil, title_given: true}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 038
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 038 --solution
```

Μια μικρή παραλλαγή

Διάβασε limit με προεπιλογή 10 από Hash που περιέχει limit: 0.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `0`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

039 / ΜΑΘΗΜΑ

Κλειδί και τιμή στο block

Έννοια: `Hash#each`, `keys`, `values`, `values_at`

Πριν αρχίσεις: 012, 013, 037

Η Hash#each δίνει κλειδί και τιμή στο block. Η γνωστή map χρησιμοποιεί επίσης αυτές τις δύο παραμέτρους πάνω σε Hash και επιστρέφει Array. Η σειρά είναι η σειρά εισαγωγής κλειδιών. Οι `keys` και `values` δίνουν Arrays, ενώ η `values_at(:a, :b)` δίνει τιμές με τη ζητούμενη σειρά και nil για κλειδιά που λείπουν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
prices = {tea: 2, cake: 4}
[prices.keys, prices.values_at(:cake, :water)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `price_lines(prices)`. Για Hash κωδικών προς ακέραιες τιμές επίστρεψε Array `CODE: PRICE` με τη σειρά εισαγωγής. Χρησιμοποίησε each με δύο παραμέτρους, χωρίς μεταβολή εισόδου.

```ruby
price_lines({tea: 2, cake: 4})
# => ["tea: 2", "cake: 4"]
```

```ruby
price_lines({})
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 039
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 039 --solution
```

Μια μικρή παραλλαγή

Πάρε τις τιμές με σειρά tea, cake, ακόμη κι αν τα κλειδιά μπήκαν ανάποδα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

040 / ΜΑΘΗΜΑ

Δεδομένα μέσα σε δεδομένα

Έννοια: Nested Hash/Array και `dig`

Πριν αρχίσεις: 006, 037, 038

Η `dig` ακολουθεί διαδοχικά κλειδιά Hash ή δείκτες Array και επιστρέφει nil όταν λείπει ενδιάμεση τιμή. Δεν μετατρέπει λανθασμένους τύπους σε Hash: αν βρει π.χ. αριθμό εκεί που θέλεις να συνεχίσεις, μπορεί να αποτύχει.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
{orders: [{city: "Athens"}]}.dig(:orders, 0, :city)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `shipping_city(order)`. Διάβασε order[:shipping][:address][:city] με dig. Ενδιάμεσες τιμές είναι Hash, nil ή απούσες. Επίστρεψε "unknown" μόνο αν το τελικό αποτέλεσμα είναι nil. Η πόλη είναι String ή nil, το κενό String διατηρείται.

RUBY
```ruby
shipping_city({shipping: {address: {city: "Patra"}}})
# => "Patra"
```

RUBY
```ruby
shipping_city({shipping: nil})
# => "unknown"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 040
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 040 --solution
```

Μια μικρή παραλλαγή

Διάβασε την πρώτη ετικέτα από εγγραφή με κενό Array labels.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `nil`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

041 / ΜΑΘΗΜΑ

Συγχώνευση ρυθμίσεων

Έννοια: `merge` και κανόνες επικράτησης

Πριν αρχίσεις: 037, 038

Η `merge` επιστρέφει νέο Hash. Σε κοινό κλειδί επικρατεί η δεξιά πλευρά, ακόμη κι αν η τιμή είναι nil ή false. Η συγχώνευση είναι ρηχή: ένα nested Hash αντικαθίσταται ως τιμή, δεν συγχωνεύεται αυτόματα σε βάθος.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
{color: "blue", active: true}.merge({active: false})
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `apply_options(defaults, options)`. Επίστρεψε νέο Hash με τις τιμές options πάνω από defaults. Μην αλλάξεις κανένα αρχικό Hash. Δεν απαιτείται βαθιά συγχώνευση ή αντιγραφή.

```ruby
apply_options({a: true, b: 2}, {a: false, b: nil})
# => {a: false, b: nil}
```

```ruby
apply_options({a: 1}, {b: 2})
# => {a: 1, b: 2}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 041
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 041 --solution
```

Μια μικρή παραλλαγή

Πρόβλεψε συγχώνευση δύο nested ρυθμίσεων.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{view: {size: 3}}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

042 / ΜΑΘΗΜΑ

Προεπιλογές εγγραφής · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 038, 041

Πρώτα όρισε defaults, μετά εφαρμογή χρήστη και τέλος περιγραφή. Η προεπιλογή αφορά απούσα πληροφορία, ενώ το false μπορεί να είναι ολοκληρωμένη επιλογή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
{enabled: true}.merge({enabled: false}).fetch(:enabled)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `booking_description(record, defaults)`. Το defaults περιέχει name String και active boolean. Το record μπορεί να τα παραλείπει ή να τα αντικαθιστά με ίδιους τύπους. Επίστρεψε `NAME: active` ή `NAME: inactive`. Κράτησε τα δύο Hashes ίδια.

```ruby
booking_description({active: false}, {name: "Eva", active: true})
# => "Eva: inactive"
```

```ruby
booking_description({}, {name: "Jo", active: true})
# => "Jo: active"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 042
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 042 --solution
```

Μια μικρή παραλλαγή

Διάβασε όνομα Guest όταν λείπει, αλλά κράτησε ρητό κενό String.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `""`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

043 / ΜΑΘΗΜΑ

Μετατροπή κλειδιών

Έννοια: `transform_keys`

Πριν αρχίσεις: 025, 037, 041

Η `transform_keys` καλεί block για κάθε κλειδί και κρατά τις τιμές. Επιστρέφει νέο Hash. Αν δύο κλειδιά γίνουν ίδια, η μεταγενέστερη εγγραφή αντικαθιστά την προηγούμενη.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
{"A" => 2, "B" => 4}.transform_keys { |key| key.downcase }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `normalized_keys(record)`. Τα κλειδιά είναι ASCII Strings. Καθάρισε άκρα και κάνε πεζά, αφήνοντας τιμές ίδιες. Σε σύγκρουση κράτησε την τελευταία τιμή. Μην αλλάξεις το αρχικό Hash.

```ruby
normalized_keys({" NAME " => "EVA", "AGE" => 2})
# => {"name" => "EVA", "age" => 2}
```

```ruby
normalized_keys({"A" => 1, "a" => 2})
# => {"a" => 2}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 043
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 043 --solution
```

Μια μικρή παραλλαγή

Κάνε μόνο κεφαλαία τα κλειδιά μιας μικρής εγγραφής.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{"X" => 0}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

044 / ΜΑΘΗΜΑ

Μετατροπή τιμών

Έννοια: `transform_values`

Πριν αρχίσεις: 037, 043

Η `transform_values` δίνει κάθε τιμή στο block και κρατά τα κλειδιά. Η τελική τιμή του block γίνεται η νέα τιμή στο νέο Hash. Χρησιμεύει για ομοιόμορφη μεταβολή ενός τιμοκαταλόγου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
{tea: 2, cake: 4}.transform_values { |price| price + 1 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `adjust_prices(prices, increase)`. Πρόσθεσε τον ακέραιο increase σε κάθε ακέραιη τιμή. Κράτησε κλειδιά και αρχικό Hash ίδια. Το αποτέλεσμα είναι νέο Hash.

```ruby
adjust_prices({a: 0, b: 5}, 2)
# => {a: 2, b: 7}
```

```ruby
adjust_prices({a: 5}, -1)
# => {a: 4}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 044
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 044 --solution
```

Μια μικρή παραλλαγή

Διπλασίασε ποσότητες στο Hash {a: 2, b: 0}.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{a: 4, b: 0}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

045 / ΜΑΘΗΜΑ

Επιλογή πεδίων

Έννοια: `slice`, `except` και `delete`

Πριν αρχίσεις: 035, 037

Η Hash#slice κρατά τα ζητούμενα κλειδιά και η except τα εξαιρεί, σε νέο Hash. Η delete αφαιρεί ένα κλειδί από το ίδιο Hash και επιστρέφει την αφαιρεμένη τιμή ή nil. Άγνωστα κλειδιά στη slice απλώς παραλείπονται.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
{name: "Eva", seats: 2, note: "x"}.slice(:name, :seats)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `public_record(record)`. Επίστρεψε νέο Hash μόνο με :name και :seats, αν υπάρχουν. Άφησε τις τιμές όπως είναι και μην αλλάξεις την αρχική εγγραφή.

RUBY
```ruby
public_record({name: "Jo", seats: 2, note: "x"})
# => {name: "Jo", seats: 2}
```

RUBY
```ruby
public_record({name: nil})
# => {name: nil}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 045
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 045 --solution
```

Μια μικρή παραλλαγή

Σε αντίγραφο εγγραφής αφαίρεσε note με delete και δώσε [αφαιρεμένη τιμή, αντίγραφο, αρχικό].

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["x", {name: "A"}, {name: "A", note: "x"}]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

046 / ΜΑΘΗΜΑ

Λίστα παραγγελιών · Εμπέδωση

Έννοια: Σύνθεση Hash, `find`, `select`, `reject` και `map`

Πριν αρχίσεις: 014, 016, 018, 037

Πάνω σε εγγραφές, το block πρώτα διαβάζει το σχετικό πεδίο. Κράτησε χωριστά τα ενεργά δεδομένα και τις ετικέτες τους, ώστε το find να επιστρέφει ολόκληρη εγγραφή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[{id: 1, ready: false}, {id: 2, ready: true}].find { |r| r[:ready] }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `order_overview(orders)`. Κάθε Hash έχει id ακέραιο, cancelled boolean, ready boolean. Αφαίρεσε ακυρωμένα. Δώσε {first_ready: πρώτη έτοιμη εγγραφή ή nil, labels: ετικέτες "Order ID" όλων των μη ακυρωμένων}. Διατήρησε είσοδο και σειρά.

```ruby
order_overview([{id: 1, cancelled: true, ready: true}, {id: 2, cancelled: false,
ready: false}])
# => {first_ready: nil, labels: ["Order 2"]}
```

```ruby
order_overview([{id: 3, cancelled: false, ready: true}])
# => {first_ready: {id: 3, cancelled: false, ready: true}, labels: ["Order 3"]}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 046
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 046 --solution
```

Μια μικρή παραλλαγή

Κράτησε μόνο ready εγγραφές και δώσε τα ids τους.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

047 / ΜΑΘΗΜΑ

Προαιρετικά ορίσματα

Έννοια: Default positional arguments

Πριν αρχίσεις: 004, 003

Το `prefix = "ID"` στον ορισμό παραμέτρου δίνει προεπιλογή όταν λείπει το όρισμα. Μια ρητή τιμή nil δεν σημαίνει ότι παραλείφθηκε. Τα positional ορίσματα αντιστοιχίζονται βάσει θέσης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def greeting(name, prefix = "Hi")
  "#{prefix} #{name}"
end
greeting("Eva")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `item_label(code, prefix = "ID")`. Επίστρεψε `PREFIX:CODE` χωρίς κενά. code και prefix είναι Strings. Αν λείπει prefix χρησιμοποίησε "ID". Κενό prefix παραμένει κενό.

```ruby
item_label("A2")
# => "ID:A2"
```

```ruby
item_label("A2", "BOX")
# => "BOX:A2"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 047
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 047 --solution
```

Μια μικρή παραλλαγή

Γράψε χαιρετισμό με default "Hello" και ρητό prefix "Hey".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Hey Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

048 / ΜΑΘΗΜΑ

Ονομασμένα ορίσματα

Έννοια: Required και optional keyword arguments

Πριν αρχίσεις: 047, 037

Το `weight:` είναι υποχρεωτικό keyword και το `remote: false` προαιρετικό. Στην κλήση γράφεις τα ονόματα, με όποια σειρά θέλεις. Το να παραλείψεις απαιτούμενο keyword προκαλεί ArgumentError. Τα keywords δίνουν νόημα σε επιλογές που αλλιώς θα ήταν διαδοχικά booleans.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def delivery(weight:, remote: false)
  base = if remote
    7
  else
    3
  end
  weight + base
end
delivery(remote: true, weight: 2)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `delivery_quote(weight:, remote: false, pickup: false)`. Το weight είναι μη αρνητικός ακέραιος. Για pickup true δώσε 0. Αλλιώς βάση 7 αν remote, 3 διαφορετικά, συν weight. Remote και pickup είναι booleans.

RUBY
```ruby
delivery_quote(weight: 2)
# => 5
```

RUBY
```ruby
delivery_quote(remote: true, weight: 2)
# => 9
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 048
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 048 --solution
```

Μια μικρή παραλλαγή

Φτιάξε formatter με υποχρεωτικό name: και προαιρετικό suffix: "!".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva?"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

049 / ΜΑΘΗΜΑ

Όρια ορατότητας

Έννοια: Method scope, block scope και block-local variables

Πριν αρχίσεις: 004, 013

Μια μέθοδος δεν βλέπει τις τοπικές μεταβλητές του καλούντος. Ένα block βλέπει τις εξωτερικές τοπικές μεταβλητές του και μπορεί να τις αλλάξει. Με `|item; temp|` το temp είναι νέα block-local μεταβλητή και δεν πειράζει εξωτερικό ομώνυμο temp.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
label = "outside"
[1].each do |n; label|
  label = "inside #{n}"
end
label
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `scoped_labels(numbers)`. Δώσε [outer_label, labels]. Το outer_label ξεκινά "original" και πρέπει να παραμείνει έτσι. Στο each χρησιμοποίησε block-local label για Strings "N!" που συγκεντρώνονται σε νέο Array. Η είσοδος είναι Array ακεραίων.

RUBY
```ruby
scoped_labels([2, 3])
# => ["original", ["2!", "3!"]]
```

RUBY
```ruby
scoped_labels([])
# => ["original", []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 049
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 049 --solution
```

Μια μικρή παραλλαγή

Με block άλλαξε ρητά εξωτερικό total από 0 σε άθροισμα 2 και 3.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

050 / ΜΑΘΗΜΑ

Προαιρετικός receiver

Έννοια: Safe navigation `&.`

Πριν αρχίσεις: 006, 024, 025

Η `&.` καλεί μέθοδο εκτός αν ο receiver είναι nil, οπότε επιστρέφει nil. Δεν παραλείπει το false και δεν κρύβει ανύπαρκτες μεθόδους. Σε αλυσίδα κάθε πιθανώς nil receiver χρειάζεται τη δική του προστασία.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
name = nil
name&.strip&.upcase
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `optional_badge(name)`. Δέχεται String ή nil. Επίστρεψε καθαρό κεφαλαίο String ή nil αντίστοιχα, με &.. Κενό String παραμένει κενό. Μην αλλάξεις το αρχικό String.

```ruby
optional_badge(" jo ")
# => "JO"
```

```ruby
optional_badge(nil)
# => nil
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 050
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 050 --solution
```

Μια μικρή παραλλαγή

Μετά την ασφαλή μετατροπή, δώσε fallback "Guest" για nil.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Guest"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

051 / ΜΑΘΗΜΑ

Ανάθεση υπό συνθήκη

Έννοια: `||=` και `&&=`

Πριν αρχίσεις: 006, 007, 037

Η `x ||= value` αναθέτει όταν το x είναι nil ή false. Η `x &&= value` αναθέτει όταν το x είναι truthy. Δεν είναι συντομογραφία ελέγχου παρουσίας κλειδιού. Χρησιμοποίησέ τες όταν ακριβώς αυτή είναι η σημασία που θέλεις.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
flag = false
flag ||= true
flag
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `fill_label(settings)`. Εδώ επιτρέπεται μεταβολή του settings Hash. Θέσε :label σε "Guest" μόνο όταν η τιμή είναι nil ή false. Επίστρεψε το ίδιο Hash. Κράτησε 0 και κενό String.

```ruby
fill_label({})
# => {label: "Guest"}
```

```ruby
fill_label({label: false})
# => {label: "Guest"}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 051
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 051 --solution
```

Μια μικρή παραλλαγή

Με &&= κάνε κεφαλαίο ένα προαιρετικό String όταν υπάρχει.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"GUEST"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

052 / ΜΑΘΗΜΑ

Μικρές διορθώσεις syntax · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 038, 049, 050, 051

Επανέλεγξε χωριστά δύο ζητήματα: λείπει το πεδίο ή λείπει ο receiver; Το Hash μπορεί να περιέχει έγκυρο false, ενώ ένα προαιρετικό όνομα μπορεί να είναι nil.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
record = {active: false, name: nil}
[record.fetch(:active, true), record[:name]&.upcase]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `safe_summary(record)`. Δώσε {active: ..., name: ...}. Active default true μόνο για απόν κλειδί, με ρητό false ή nil διατηρημένο. Το name είναι String, nil ή απόν: καθάρισε/κεφαλαιοποίησε όταν υπάρχει, αλλιώς δώσε "Guest". Μην μεταβάλεις record.

```ruby
safe_summary({active: false, name: nil})
# => {active: false, name: "Guest"}
```

```ruby
safe_summary({name: " eva "})
# => {active: true, name: "EVA"}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 052
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 052 --solution
```

Μια μικρή παραλλαγή

Διατήρησε ρητό nil σε enabled με fetch και default true.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `nil`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

053 / ΜΑΘΗΜΑ

Ταξινόμηση έτοιμων τιμών

Έννοια: `sort` και `reverse`

Πριν αρχίσεις: 011, 035

Η `sort` δίνει νέο ταξινομημένο Array και η `reverse` νέο Array με ανάποδη σειρά. Τα αριθμητικά στοιχεία ταξινομούνται ως αριθμοί, τα Strings σύμφωνα με τη σύγκρισή τους. Εδώ χρησιμοποιούμε ASCII κωδικούς, χωρίς κανόνες ανθρώπινης αλφαβητικής ταξινόμησης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["B2", "A1", "B1"].sort.reverse
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `code_orders(codes)`. Για ASCII Strings επίστρεψε [ascending, descending]. Διατήρησε διπλότυπα και αρχική λίστα. Και τα δύο αποτελέσματα είναι νέα Arrays.

```ruby
code_orders(["B", "A", "B"])
# => [["A", "B", "B"], ["B", "B", "A"]]
```

```ruby
code_orders([])
# => [[], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 053
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 053 --solution
```

Μια μικρή παραλλαγή

Ταξινόμησε τους ακεραίους 10, 2, 1 και σύγκρινε με την ίδια λίστα ως Strings.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[[1, 2, 10], ["1", "10", "2"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

054 / ΜΑΘΗΜΑ

Ταξινόμηση με κλειδί

Έννοια: `sort_by` και σύνθετο κλειδί

Πριν αρχίσεις: 037, 053

Η `sort_by` ζητά από το block ένα κλειδί σύγκρισης για κάθε στοιχείο και επιστρέφει τις ίδιες εγγραφές σε νέο Array. Με Array κλειδί συγκρίνεται πρώτα το πρώτο πεδίο και σε ισοπαλία το επόμενο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["bbbb", "a", "cc"].sort_by { |s| s.length }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `ordered_tasks(tasks)`. Κάθε Hash έχει priority ακέραιο και name ASCII String. Ταξινόμησε αύξουσα priority και μετά αύξον name. Δεν απαιτείται ιδιαίτερη σειρά για πλήρως ίσα κλειδιά. Μην αλλάξεις την είσοδο.

```ruby
ordered_tasks([{priority: 2, name: "A"}, {priority: 1, name: "B"}, {priority: 1,
name: "A"}])
# => [{priority: 1, name: "A"}, {priority: 1, name: "B"}, {priority: 2, name: "A"}]
```

```ruby
ordered_tasks([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 054
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 054 --solution
```

Μια μικρή παραλλαγή

Ταξινόμησε ονόματα πρώτα κατά μήκος και μετά αλφαβητικά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Al", "Jo", "Eva"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

055 / ΜΑΘΗΜΑ

Μικρότερο και μεγαλύτερο

Έννοια: `min`, `max`, `min_by`, `max_by`

Πριν αρχίσεις: 015, 037, 054

Οι min/max δίνουν μικρότερη/μεγαλύτερη τιμή. Οι min_by/max_by επιστρέφουν το αρχικό στοιχείο που έχει το αντίστοιχο κλειδί. Σε κενή συλλογή δίνουν nil. Μπορείς να βάλεις δευτερεύον κλειδί για ρητή επίλυση ισοπαλίας.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
[{name: "A", price: 5}, {name: "B", price: 2}].min_by { |p| p[:price] }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `cheapest_product(products)`. Κάθε εγγραφή έχει code ASCII String, price μη αρνητικό ακέραιο και available boolean. Δώσε ολόκληρη φθηνότερη διαθέσιμη εγγραφή ή nil. Σε ίδια τιμή, προτίμησε μικρότερο code.

RUBY
```ruby
cheapest_product([{code: "A", price: 1, available: false}, {code: "B", price: 4,
available: true}])
# => {code: "B", price: 4, available: true}
```

RUBY
```ruby
cheapest_product([{code: "B", price: 4, available: true}, {code: "A", price: 4,
available: true}])
# => {code: "A", price: 4, available: true}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 055
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 055 --solution
```

Μια μικρή παραλλαγή

Δώσε μόνο μικρότερη και μεγαλύτερη αριθμητική τιμή από [8, 2, 5].

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 8]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

056 / ΜΑΘΗΜΑ

Μία εμφάνιση ανά ταυτότητα

Έννοια: `uniq` με και χωρίς block

Πριν αρχίσεις: 037, 014

Η `uniq` κρατά την πρώτη εμφάνιση κάθε ίσης τιμής. Με block χρησιμοποιεί το αποτέλεσμα ως ταυτότητα, αλλά επιστρέφει το αρχικό στοιχείο. Η σειρά των πρώτων εμφανίσεων διατηρείται, χωρίς ταξινόμηση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["A", "B", "A"].uniq
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `first_products(products)`. Κάθε Hash έχει :code. Κράτησε την πρώτη εγγραφή ανά code σε νέο Array. Οι υπόλοιπες τιμές μπορεί να διαφέρουν. Μην αλλάξεις τις εγγραφές.

```ruby
first_products([{code: "B", price: 2}, {code: "A", price: 3}, {code: "B", price: 9}])
# => [{code: "B", price: 2}, {code: "A", price: 3}]
```

```ruby
first_products([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 056
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 056 --solution
```

Μια μικρή παραλλαγή

Κράτησε ένα όνομα ανά πεζή μορφή χωρίς να αλλάξεις τη γραφή της πρώτης εμφάνισης.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Eva", "Jo"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

057 / ΜΑΘΗΜΑ

Αφαίρεση απουσίας

Έννοια: `compact`

Πριν αρχίσεις: 006, 016

Η `compact` αφαιρεί μόνο nil και επιστρέφει νέο Array. Κρατά false, μηδέν και κενό String. Χρησιμεύει για προαιρετικά αποτελέσματα στα οποία το false παραμένει χρήσιμο δεδομένο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[nil, false, 0, ""].compact
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `present_values(values)`. Επίστρεψε νέο Array χωρίς nil. Κράτησε όλες τις άλλες τιμές, τη σειρά και τα διπλότυπα. Η αρχική είσοδος μένει ίδια.

```ruby
present_values([nil, false, 0, "", nil, false])
# => [false, 0, "", false]
```

```ruby
present_values([nil, nil])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 057
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 057 --solution
```

Μια μικρή παραλλαγή

Καθάρισε προαιρετικά ονόματα από nil και ένωσε όσα έμειναν.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva, Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

058 / ΜΑΘΗΜΑ

Ομάδες με κοινό γνώρισμα

Έννοια: `group_by`

Πριν αρχίσεις: 037, 039

Η `group_by` υπολογίζει κλειδί ανά στοιχείο και επιστρέφει Hash: κάθε κλειδί έχει Array αρχικών στοιχείων. Η σειρά μέσα στην ομάδα διατηρείται. Η ομαδοποίηση δεν ταξινομεί και δεν αντιγράφει τις ίδιες τις εγγραφές.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["a", "bb", "c"].group_by { |s| s.length }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `products_by_category(products)`. Κάθε Hash έχει :category String. Ομαδοποίησε τις εγγραφές ανά κατηγορία. Κράτησε σειρά μέσα στην ομάδα και αρχικές τιμές. Κενή είσοδος δίνει {}.

```ruby
products_by_category([{category: "food", id: 1}, {category: "book", id: 2},
{category: "food", id: 3}])
# => {"food" => [{category: "food", id: 1}, {category: "food", id: 3}], "book" =>
[{category: "book", id: 2}]}
```

```ruby
products_by_category([])
# => {}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 058
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 058 --solution
```

Μια μικρή παραλλαγή

Ομαδοποίησε ποσότητες σε true/false ανάλογα αν είναι τουλάχιστον 5.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{false => [2], true => [5, 8]}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

059 / ΜΑΘΗΜΑ

Συχνότητες τιμών

Έννοια: `tally`

Πριν αρχίσεις: 037, 039

Η `tally` μετρά εμφανίσεις και δίνει Hash τιμή προς πλήθος. Δεν παίρνει block μετατροπής. Αν θέλεις κανονικοποίηση πριν τη μέτρηση, κάνε map και μετά tally.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["tea", "cake", "tea"].tally
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `choice_counts(choices)`. Για Array ASCII Strings, καθάρισε άκρα και κάνε πεζά, μετά μέτρησε εμφανίσεις. Το κενό String παραμένει κανονική επιλογή. Μην αλλάξεις την είσοδο.

```ruby
choice_counts([" Tea ", "TEA", "cake"])
# => {"tea" => 2, "cake" => 1}
```

```ruby
choice_counts(["", " "])
# => {"" => 2}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 059
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 059 --solution
```

Μια μικρή παραλλαγή

Μέτρησε έτοιμες boolean επιλογές, κρατώντας και false.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{true => 2, false => 1}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

060 / ΜΑΘΗΜΑ

Μικρές ομάδες προϊόντων · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 044, 056, 058

Ξεκαθάρισε σε ποιο στάδιο εφαρμόζεται η μοναδικότητα. Εδώ ένα code αντιστοιχεί σε ένα προϊόν, ανεξάρτητα από την κατηγορία. Πρώτα κρατάς την πρώτη εγγραφή και μετά συνοψίζεις.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[{code: "A", group: "x"}, {code: "A", group: "y"}].uniq { |p| p[:code] }.group_by {
|p| p[:group] }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `category_counts(products)`. Κάθε εγγραφή έχει code και category Strings. Κράτησε πρώτη ανά code και δώσε Hash κατηγορίας προς πλήθος μοναδικών προϊόντων. Κράτησε την είσοδο ίδια.

```ruby
category_counts([{code: "A", category: "x"}, {code: "A", category: "y"}, {code: "B",
category: "x"}])
# => {"x" => 2}
```

```ruby
category_counts([])
# => {}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 060
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 060 --solution
```

Μια μικρή παραλλαγή

Αντί για πλήθη, δώσε τους κωδικούς κάθε ομάδας.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{"x" => ["A", "B"]}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

061 / ΜΑΘΗΜΑ

Δύο συμπληρωματικές ομάδες

Έννοια: `partition`

Πριν αρχίσεις: 015, 016, 037

Η `partition` χωρίζει σε δύο Arrays: πρώτα όσα δίνουν truthy στο block και μετά τα υπόλοιπα. Επιστρέφει Array δύο στοιχείων. Κρατά τη σειρά μέσα σε καθεμία από τις ομάδες.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[2, 7, 1].partition { |n| n >= 5 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `readiness_groups(records)`. Κάθε Hash έχει ready boolean. Δώσε [ready_records, pending_records] με partition, χωρίς μεταβολή της εισόδου.

```ruby
readiness_groups([{id: 1, ready: false}, {id: 2, ready: true}])
# => [[{id: 2, ready: true}], [{id: 1, ready: false}]]
```

```ruby
readiness_groups([])
# => [[], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 061
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 061 --solution
```

Μια μικρή παραλλαγή

Χώρισε ονόματα σε κενά και μη κενά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[[""], ["A", "B"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

062 / ΜΑΘΗΜΑ

Μετατροπή μαζί με απόρριψη

Έννοια: `filter_map`

Πριν αρχίσεις: 006, 014, 015

Η `filter_map` μετατρέπει κάθε στοιχείο και κρατά μόνο truthy επιστροφές. Αφαιρεί nil και false, αλλά κρατά 0 και κενό String. Μια συνθήκη χωρίς else επιστρέφει nil όταν δεν ισχύει και ταιριάζει σε αυτόν τον μηχανισμό.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[1, 3, 2].filter_map { |n| "N#{n}" if n >= 2 }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `ready_labels(records)`. Κάθε Hash έχει id ακέραιο και ready boolean. Δώσε Strings "Ready ID" μόνο για τις ready εγγραφές, με filter_map. Κράτησε σειρά και αρχικά δεδομένα.

```ruby
ready_labels([{id: 1, ready: false}, {id: 0, ready: true}])
# => ["Ready 0"]
```

```ruby
ready_labels([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 062
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 062 --solution
```

Μια μικρή παραλλαγή

Με filter_map επίστρεψε την ίδια τιμή από [nil, false, 0, ""]. Πρόβλεψε τι μένει.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[0, ""]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

063 / ΜΑΘΗΜΑ

Πολλά αποτελέσματα από κάθε στοιχείο

Έννοια: `flat_map` και ένα επίπεδο ένωσης

Πριν αρχίσεις: 014, 037

Η `flat_map` περιμένει ότι ένα στοιχείο μπορεί να παράγει πολλές τιμές. Συγκεντρώνει τις επιστροφές και ενώνει ένα επίπεδο Arrays. Δεν κάνει απεριόριστο flatten και δεν αφαιρεί διπλότυπα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[1, 2].flat_map { |n| [n, n * 10] }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `all_tags(products)`. Κάθε Hash έχει tags Array Strings. Δώσε όλες τις ετικέτες σε ένα Array, με τη σειρά προϊόντων και ετικετών. Κράτησε επαναλήψεις και είσοδο ίδια.

```ruby
all_tags([{tags: ["a", "b"]}, {tags: []}, {tags: ["a"]}])
# => ["a", "b", "a"]
```

```ruby
all_tags([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 063
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 063 --solution
```

Μια μικρή παραλλαγή

Δώσε δύο labels για κάθε όνομα: κανονικό και κεφαλαίο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Eva", "EVA", "Jo", "JO"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

064 / ΜΑΘΗΜΑ

Παρατήρηση μέσα σε αλυσίδα

Νέα έννοια: `tap` και επιστροφή του receiver.

Πριν αρχίσεις: χρειάζονται Array, `<<`, blocks, `map`, `reject`, `strip`, `empty?`, `sort`, `join` και η διάκριση αναφοράς από αντίγραφο. Αυτά καλύπτονται στις προηγούμενες ενότητες 011-063.

Μια αλυσίδα κλήσεων μεταφέρει το αποτέλεσμα κάθε βήματος στο επόμενο. Μερικές φορές θέλεις να παρατηρήσεις μια ενδιάμεση τιμή. Η `tap` δίνει ολόκληρο τον receiver στο block και μετά επιστρέφει τον ίδιο receiver.

RUBY
```ruby
values = [3, 8]
audit = []

result = values.tap do |items|
  audit << items.length
  "μια άλλη τιμή"
end

p result
# => [3, 8]

p audit
# => [2]
```

Το block εκτελείται μία φορά με όλο το Array. Η `items` αναφέρεται στο ίδιο αντικείμενο με τη `values`. Καταγράφουμε το πλήθος σε άλλη λίστα.

Η τελευταία έκφραση του block είναι το String `"μια άλλη τιμή"`. Παρ' όλα αυτά, η `tap` επιστρέφει το αρχικό Array. Δεν χρησιμοποιεί την επιστροφή του block ως δική της επιστροφή. Αυτή είναι η σύμβαση της Kernel#tap.

Πρόσεξε τη διαφορά ανάμεσα σε «ίδιο αντικείμενο» και «αντικείμενο με ίδιο περιεχόμενο». Η `tap` δεν δημιουργεί αντίγραφο και δεν προστατεύει τον receiver από αλλαγές μέσα στο block.

Πρόβλεψε το αποτέλεσμα.

RUBY
```ruby
values = [3, 8]

result = values.tap do |items|
  items << 12
end
```

Και η `values` και η `result` αναφέρονται στο ίδιο Array, με περιεχόμενο `[3, 8, 12]`. Η προσθήκη έγινε στο ίδιο το αντικείμενο.

Η άσκηση

Η `name_report(names, audit)` έχει ήδη μια αλυσίδα που:

1. Καθαρίζει τα άκρα κάθε ονόματος.
2. Αφαιρεί τα κενά αποτελέσματα.
3. Ταξινομεί τα ονόματα.
4. Τα ενώνει με `", "`.

Πρόσθεσε **μία κλήση** `tap` **μετά τη** `reject` **και πριν από τη** `sort`. Το block της θα προσθέτει στο `audit` την ενδιάμεση λίστα καθαρών, μη κενών ονομάτων, στη σειρά που είχαν πριν από την ταξινόμηση.

Η μέθοδος πρέπει να συνεχίσει να επιστρέφει το τελικό κείμενο. Το `names` και τα αρχικά του Strings πρέπει να μείνουν ίδια. Στο `audit` επιτρέπεται ακριβώς μία νέα εγγραφή, διατηρώντας τις παλιές του.

Το `names` είναι Array από Strings. Το `audit` είναι ξεχωριστό Array και δεν μοιράζεται στοιχεία με το `names`. Χρησιμοποιούμε τη συνηθισμένη σειρά της Ruby `sort`, χωρίς κανόνες λεξικογραφικής ταξινόμησης ανά γλώσσα.

```ruby
names = [" Nikos ", " ", "Anna", " Maria "]
audit = []

result = name_report(names, audit)

# result => "Anna, Maria, Nikos"
# audit  => [["Nikos", "Anna", "Maria"]]
# names  => [" Nikos ", " ", "Anna", " Maria "]
```

Για κενή είσοδο, η μέθοδος επιστρέφει κενό String και προσθέτει κενό Array στο audit.

```ruby
audit = []
result = name_report([], audit)

# result => ""
# audit  => [[]]
```

Γράψε στο exercise.rb. Η αλυσίδα είναι ήδη δοσμένη. Από τον φάκελο `syntax-first/ruby`.

```sh
bin/lesson 064
```

Κάποια tests περνούν ήδη, επειδή η αρχική αλυσίδα παράγει το σωστό τελικό κείμενο. Εκείνα που ελέγχουν το audit θα αποτύχουν μέχρι να προσθέσεις το ζητούμενο βήμα.

Μια μικρή παραλλαγή

Σε δεύτερη μέθοδο `name_report_with_counts(names, audit)`, πρόσθεσε δύο κλήσεις `tap`: μία πριν από το καθάρισμα και μία μετά την απόρριψη κενών. Αυτή τη φορά κατέγραψε μόνο τα δύο πλήθη.

```ruby
audit = []
result = name_report_with_counts([" Nikos ", " ", "Anna"], audit)

# result => "Anna, Nikos"
# audit  => [3, 2]
```

Η παραλλαγή χρησιμοποιεί τα ίδια εργαλεία και είναι χωριστή από την κύρια προσπάθεια και τα tests της.

Αν κολλήσεις

Το block της `tap` παίρνει ολόκληρη την ενδιάμεση λίστα. Μία προσθήκη με `<<` αρκεί. Δεν χρειάζεται επιπλέον `each` ούτε να επιστρέψεις χειροκίνητα την ενδιάμεση λίστα από το block.

065 / ΜΑΘΗΜΑ

Συνέχεια με το αποτέλεσμα του block

Έννοια: `then` / `yield_self`

Πριν αρχίσεις: 014, 016, 024, 064

Η `then`, με εναλλακτικό όνομα `yield_self`, δίνει ολόκληρο τον receiver στο block και επιστρέφει την τελική τιμή του block. Μπορεί λοιπόν να αλλάξει τον τύπο αποτελέσματος μιας αλυσίδας. Η tap επιστρέφει τον receiver και αγνοεί την επιστροφή του block.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[2, 3].then { |numbers| "Total: #{numbers.sum}" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `names_summary(names)`. Καθάρισε String ονόματα με strip και αφαίρεσε κενά. Με then δώσε Hash {count: πλήθος, text: ονόματα με ", "}. Κράτησε σειρά και είσοδο.

```ruby
names_summary([" Eva ", " ", "Jo"])
# => {count: 2, text: "Eva, Jo"}
```

```ruby
names_summary([])
# => {count: 0, text: ""}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 065
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 065 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε yield_self για να βάλεις αγκύλες γύρω από ήδη ενωμένο κείμενο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"[A, B]"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

066 / ΜΑΘΗΜΑ

Ποια μέθοδος ταιριάζει; · Εμπέδωση

Έννοια: Επιλογή μεταξύ μετατροπής, φίλτρου, εύρεσης και σύνοψης

Πριν αρχίσεις: 014, 015, 018, 021, 037

Η μορφή της ζητούμενης απάντησης καθοδηγεί την επιλογή: Array μετασχηματισμών, υποσύνολο, μία εγγραφή ή αριθμός. Δοκίμασε να ονομάσεις αυτά τα τέσσερα αποτελέσματα πριν γράψεις κώδικα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = [1, 4, 2]
{first: values.find { |n| n > 2 }, total: values.sum}
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `stock_requests(stocks)`. Οι ποσότητες είναι μη αρνητικοί ακέραιοι. Δώσε Hash με labels "N τεμ." για καθεμία, positive μόνο θετικές, first_large την πρώτη >=5 ή nil και total άθροισμα όλων.

```ruby
stock_requests([0, 7, 2])
# => {labels: ["0 τεμ.", "7 τεμ.", "2 τεμ."], positive: [7, 2], first_large: 7,
total: 9}
```

```ruby
stock_requests([])
# => {labels: [], positive: [], first_large: nil, total: 0}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 066
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 066 --solution
```

Μια μικρή παραλλαγή

Βρες πόσες ποσότητες είναι τουλάχιστον 5 χωρίς ενδιάμεσο Array.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `2`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

067 / ΜΑΘΗΜΑ

Αναφορά αποθήκης · Εμπέδωση

Έννοια: Σύνθεση ομαδοποίησης, ταξινόμησης και αθροίσματος

Πριν αρχίσεις: 015, 021, 039, 054, 058

Η αναφορά χτίζεται σε στάδια: κρατάς χρήσιμες εγγραφές, ομαδοποιείς, αθροίζεις και ταξινομείς τις γραμμές. Κάθε στάδιο έχει γνωστό σχήμα εξόδου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[{category: "x", quantity: 2}, {category: "x", quantity: 3}].group_by { |p|
p[:category] }.transform_values { |rows| rows.sum { |p| p[:quantity] } }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `warehouse_report(products)`. Κάθε Hash έχει category ASCII String και quantity μη αρνητικό ακέραιο. Αγνόησε quantity 0. Δώσε Array [category, total] ανά κατηγορία, ταξινομημένο αύξουσα κατηγορία. Μην αλλάξεις την είσοδο.

```ruby
warehouse_report([{category: "b", quantity: 2}, {category: "a", quantity: 3},
{category: "b", quantity: 4}, {category: "z", quantity: 0}])
# => [["a", 3], ["b", 6]]
```

```ruby
warehouse_report([{category: "a", quantity: 0}])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 067
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 067 --solution
```

Μια μικρή παραλλαγή

Μορφοποίησε ήδη έτοιμες γραμμές [["a", 3], ["b", 6]] σε κείμενο με newline.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"a: 3\nb: 6"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

068 / ΜΑΘΗΜΑ

Κατάσταση μαζί με συμπεριφορά

Έννοια: `class`, `new`, `initialize` και instance variables

Πριν αρχίσεις: 004, 037

Μια κλάση ορίζει συμπεριφορά για αντικείμενα. Η `new` δημιουργεί αντικείμενο και καλεί `initialize` με τα δοσμένα ορίσματα. Οι μεταβλητές `@name` ανήκουν στο συγκεκριμένο αντικείμενο και είναι διαθέσιμες στις μεθόδους του. Η initialize προετοιμάζει την κατάσταση· η new επιστρέφει το αντικείμενο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class SeatLabel
  def initialize(number)
    @number = number
  end
  def text
    "Seat #{@number}"
  end
end
SeatLabel.new(4).text
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε κλάση Booking με initialize(name, seats) και description που επιστρέφει "NAME: SEATS θέσεις". Οι είσοδοι είναι String και μη αρνητικός ακέραιος. Κάθε αντικείμενο κρατά τη δική του κατάσταση.

RUBY
```ruby
Booking.new("Eva", 2).description
# => "Eva: 2 θέσεις"
```

RUBY
```ruby
a = Booking.new("A", 1)
b = Booking.new("B", 3)
[a.description, b.description]
# => ["A: 1 θέσεις", "B: 3 θέσεις"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 068
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 068 --solution
```

Μια μικρή παραλλαγή

Γράψε μικρή κλάση Ticket που κρατά price και quantity και δίνει total.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `12`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

069 / ΜΑΘΗΜΑ

Ανάγνωση και αλλαγή ιδιοτήτων

Έννοια: `attr_reader`, `attr_writer`, `attr_accessor`

Πριν αρχίσεις: 068

Η `attr_reader :id` δημιουργεί μέθοδο ανάγνωσης id. Η `attr_writer :status` δημιουργεί setter status=. Η `attr_accessor` δημιουργεί και τις δύο. Η πρόσβαση δεν προσθέτει έλεγχο εγκυρότητας. Η ονομασία :id είναι Symbol, όχι η ίδια η τιμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class Note
  attr_accessor :text
end
note = Note.new
note.text = "Ready"
note.text
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Reservation.new(id, status). Το id διαβάζεται αλλά δεν αλλάζει μέσω setter. Το status διαβάζεται και αλλάζει. Χρησιμοποίησε attr_reader και attr_accessor. Οι τιμές δίνονται έγκυρες.

```ruby
r = Reservation.new(7, "new")
[r.id, r.status]
# => [7, "new"]
```

```ruby
r = Reservation.new(7, "new")
r.status = "ready"
r.status
# => "ready"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 069
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 069 --solution
```

Μια μικρή παραλλαγή

Δώσε μόνο setter για note και εμφάνισέ το μέσω ξεχωριστής description.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Note: Hi"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

070 / ΜΑΘΗΜΑ

Ο τρέχων receiver

Έννοια: `self` και setter methods

Πριν αρχίσεις: 069, 009

Το `self` είναι ο τρέχων receiver. Μέσα σε μέθοδο, `quantity = value` ορίζει τοπική μεταβλητή. Το `self.quantity = value` καλεί setter. Ένας setter μπορεί να ελέγχει όρια πριν αποθηκεύσει σε @quantity. Η έκφραση ανάθεσης επιστρέφει το δεξί όρισμα, ανεξάρτητα από την επιστροφή του setter.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class Counter
  attr_accessor :value
  def reset
    self.value = 0
  end
end
c = Counter.new
c.reset
c.value
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Stock.new(quantity), quantity reader, quantity=(value) που κρατά 0 για αρνητική τιμή και την τιμή αλλιώς, και add(amount) που καλεί τον setter με quantity + amount και επιστρέφει τη νέα quantity. Όλες οι τιμές ακέραιοι.

```ruby
s = Stock.new(2)
s.add(3)
# => 5
```

```ruby
s = Stock.new(2)
s.add(-7)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 070
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 070 --solution
```

Μια μικρή παραλλαγή

Δείξε ότι μια τοπική ανάθεση με ίδιο όνομα δεν καλεί setter.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `2`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

071 / ΜΑΘΗΜΑ

Ονόματα με συμβάσεις

Έννοια: Μέθοδοι `?`, `!`, `=` και τα όρια των συμβάσεων

Πριν αρχίσεις: 068, 069, 034

Το ? συνήθως δηλώνει ερώτηση και το ! μια πιο επικίνδυνη εκδοχή, συχνά με μεταβολή. Είναι μέρος του ονόματος, χωρίς αυτόματο boolean ή αυτόματη αλλαγή. Το = επιτρέπει σύνταξη ανάθεσης και αντιστοιχεί σε setter.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class Switch
  def on?
    true
  end
end
Switch.new.on?
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Toggle.new που αρχίζει ανενεργό. Η active? δίνει boolean. Η activate! αλλάζει την κατάσταση σε true και επιστρέφει το ίδιο αντικείμενο. Επαναλαμβανόμενη ενεργοποίηση παραμένει true.

```ruby
Toggle.new.active?
# => false
```

```ruby
t = Toggle.new
t.activate!
t.active?
# => true
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 071
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 071 --solution
```

Μια μικρή παραλλαγή

Φτιάξε predicate empty? για δικό σου μικρό αντικείμενο που κρατά Array.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

072 / ΜΑΘΗΜΑ

Δημόσιο API και βοηθητικές μέθοδοι

Έννοια: `public`, `private`, `protected`

Πριν αρχίσεις: 068, 069

Οι μέθοδοι είναι συνήθως public. Μετά το `private`, οι βοηθητικές μέθοδοι καλούνται εσωτερικά, συνήθως χωρίς receiver. Οι protected επιτρέπουν συνεργασία μεταξύ συμβατών αντικειμένων από το πλαίσιο μεθόδων τους, αλλά δεν αποτελούν δημόσιο API. Το `public` επαναφέρει δημόσια ορατότητα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class WrappedText
  def text
    "[#{content}]"
  end
  private
  def content
    "OK"
  end
end
WrappedText.new.text
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Label.new(text) με public render που επιστρέφει καθαρό κεφαλαίο text μέσα σε []. Η normalized είναι private βοηθητική μέθοδος και επιστρέφει strip.upcase. Μην αλλάζεις την αρχική είσοδο.

RUBY
```ruby
Label.new(" jo ").render
# => "[JO]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 072
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 072 --solution
```

Μια μικρή παραλλαγή

Δύο αντικείμενα SecretRank συγκρίνουν εσωτερική protected rank, μέσω δημόσιας same_rank?.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

073 / ΜΑΘΗΜΑ

Μέθοδοι της κλάσης

Έννοια: Class methods και class instance variables

Πριν αρχίσεις: 068, 069

Το `def self.default_limit` ορίζει μέθοδο πάνω στην κλάση. Το `@limit` μέσα στη μέθοδο αυτή ανήκει στην κλάση ως αντικείμενο. Το `@limit` μέσα σε instance method ανήκει στην κάθε ξεχωριστή παρουσία. Ίδιο όνομα δεν σημαίνει ίδια αποθήκευση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class Defaults
  @size = 4
  def self.size
    @size
  end
end
Defaults.size
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Room με class method default_capacity που δίνει 4 και Room.new(capacity = Room.default_capacity). Κάθε room έχει reader capacity. Αποθήκευσε το default σε class instance variable @default_capacity.

RUBY
```ruby
[Room.default_capacity, Room.new.capacity]
# => [4, 4]
```

RUBY
```ruby
[Room.new(9).capacity, Room.new.capacity]
# => [9, 4]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 073
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 073 --solution
```

Μια μικρή παραλλαγή

Δώσε class method label που επιστρέφει σταθερό κείμενο για όλη την κλάση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Main hall"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

074 / ΜΑΘΗΜΑ

Ονόματα σε δικό τους χώρο

Έννοια: Constants, modules ως namespaces και `::`

Πριν αρχίσεις: 068, 073

Ένα module μπορεί να οργανώνει ονόματα χωρίς να δημιουργεί instances. Το `::` προσπελαύνει constant μέσα σε namespace. Τα ονόματα που αρχίζουν κεφαλαίο είναι constants· η κλάση είναι και η ίδια αντικείμενο που αποθηκεύεται σε constant.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
module Kitchen
  DEFAULT = "tea"
end
Kitchen::DEFAULT
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Catalog::Entry.new(name) με label "Product: NAME" και Guests::Entry.new(name) με label "Guest: NAME". Οι δύο κλάσεις έχουν ίδιο σύντομο όνομα σε διαφορετικά modules.

```ruby
Catalog::Entry.new("Tea").label
# => "Product: Tea"
```

```ruby
Guests::Entry.new("Eva").label
# => "Guest: Eva"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 074
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 074 --solution
```

Μια μικρή παραλλαγή

Όρισε DEFAULT_STATUS μέσα σε module Bookings.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"pending"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

075 / ΜΑΘΗΜΑ

Δύο μικρά αντικείμενα · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 069, 070, 073

Δώσε σε κάθε αντικείμενο τη δική του κατάσταση και χρησιμοποίησε την κλάση μόνο για κοινή προεπιλογή. Η άσκηση ελέγχει δύο παρουσίες ώστε να φανεί τυχόν κατά λάθος κοινή αποθήκευση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class TinyBox
  attr_accessor :count
  def initialize
    @count = 1
  end
end
a = TinyBox.new
b = TinyBox.new
a.count = 3
[a.count, b.count]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε SeatBooking με default_seats class method = 1. Η new(name, seats = SeatBooking.default_seats) κρατά name μόνο για ανάγνωση και seats με setter που περιορίζει αρνητικά στο 0. Δύο κρατήσεις πρέπει να είναι ανεξάρτητες.

```ruby
SeatBooking.new("A").seats
# => 1
```

```ruby
a = SeatBooking.new("A")
b = SeatBooking.new("B")
a.seats = 3
[a.seats, b.seats]
# => [3, 1]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 075
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 075 --solution
```

Μια μικρή παραλλαγή

Συγκέντρωσε ονόματα από δύο αντικείμενα με reader name.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Eva", "Jo"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

076 / ΜΑΘΗΜΑ

Αντικείμενα που συνεργάζονται

Έννοια: Composition και duck typing

Πριν αρχίσεις: 068, 014

Με composition ένα αντικείμενο κρατά έναν συνεργάτη και καλεί τη μέθοδό του. Duck typing σημαίνει ότι ο συνεργάτης χρειάζεται τη συμφωνημένη συμπεριφορά, όχι συγκεκριμένο γονικό τύπο. Εδώ αρκεί η μέθοδος format(name).

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class UpperFormatter
  def format(name)
    name.upcase
  end
end
UpperFormatter.new.format("Eva")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Report.new(formatter) με lines(names), που καλεί formatter.format(name) για κάθε String και επιστρέφει Array αποτελεσμάτων. Πρόσθεσε PlainFormatter που επιστρέφει name και BracketFormatter που επιστρέφει [NAME] ως String.

```ruby
Report.new(PlainFormatter.new).lines(["Eva", "Jo"])
# => ["Eva", "Jo"]
```

```ruby
Report.new(BracketFormatter.new).lines(["Eva"])
# => ["[Eva]"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 076
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 076 --solution
```

Μια μικρή παραλλαγή

Φτιάξε τρίτο formatter που δίνει "Hi NAME".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Hi Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

077 / ΜΑΘΗΜΑ

Εξειδίκευση υπάρχουσας συμπεριφοράς

Έννοια: Inheritance και overriding

Πριν αρχίσεις: 068, 069

Το `class Child < Parent` δίνει κληρονομικότητα. Αν η Child δεν ορίζει μια μέθοδο, αναζητείται στον γονέα. Αν ορίσει μέθοδο με το ίδιο όνομα, την αντικαθιστά για τις δικές της παρουσίες. Η κατάσταση @... εξακολουθεί να ανήκει στην παρουσία.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class BasicTag
  def label
    "basic"
  end
end
class SpecialTag < BasicTag
  def label
    "special"
  end
end
[BasicTag.new.label, SpecialTag.new.label]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε StandardBooking.new(name) με label "Booking NAME". Η VipBooking κληρονομεί αρχικοποίηση και κάνει override τη label σε "VIP NAME".

RUBY
```ruby
StandardBooking.new("Eva").label
# => "Booking Eva"
```

RUBY
```ruby
VipBooking.new("Jo").label
# => "VIP Jo"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 077
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 077 --solution
```

Μια μικρή παραλλαγή

Δείξε κληρονόμηση μεθόδου χωρίς override.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"A1"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

078 / ΜΑΘΗΜΑ

Συνέχεια στη γονική υλοποίηση

Έννοια: `super` και `super()`

Πριν αρχίσεις: 077, 047

Το σκέτο `super` καλεί τη γονική υλοποίηση της ίδιας μεθόδου με τα τρέχοντα ορίσματα. Το `super()` περνά κανένα όρισμα. Το `super(name)` προωθεί ρητά μόνο το name. Αυτό είναι χρήσιμο όταν η υποκλάση προσθέτει δικές της παραμέτρους.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class BaseGreeting
  def text
    "Hello"
  end
end
class WarmGreeting < BaseGreeting
  def text
    "#{super()}!"
  end
end
WarmGreeting.new.text
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε BasicReservation.new(name) με label name. Η NumberedReservation.new(name, number) καλεί τη γονική initialize με name, κρατά number και label "NUMBER: NAME" χρησιμοποιώντας super().

RUBY
```ruby
NumberedReservation.new("Eva", 4).label
# => "4: Eva"
```

RUBY
```ruby
BasicReservation.new("Jo").label
# => "Jo"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 078
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 078 --solution
```

Μια μικρή παραλλαγή

Κάνε override μέθοδο με ίδιο όρισμα και προώθησέ το αυτόματα με σκέτο super.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"HI"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

079 / ΜΑΘΗΜΑ

Κοινή συμπεριφορά με module

Έννοια: `include` και mixins

Πριν αρχίσεις: 068, 074

Ένα module μπορεί να ορίζει instance methods. Με `include ModuleName` μια κλάση αποκτά αυτή τη συμπεριφορά. Το module μπορεί να καλεί μεθόδους που αναμένεται να παρέχει η κλάση, όπως name. Αυτή η μικρή συμφωνία πρέπει να είναι σαφής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
module Exclamation
  def shout(text)
    "#{text}!"
  end
end
class Speaker
  include Exclamation
end
Speaker.new.shout("Hi")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε module NamedLabel με label που επιστρέφει "[#{name}]". Οι PersonLabel και ProductLabel το περιλαμβάνουν και δέχονται name στην initialize, με attr_reader :name. Καμία δεν κληρονομεί από την άλλη.

RUBY
```ruby
PersonLabel.new("Eva").label
# => "[Eva]"
```

RUBY
```ruby
ProductLabel.new("Tea").label
# => "[Tea]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 079
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 079 --solution
```

Μια μικρή παραλλαγή

Μοιράσου predicate positive? που βασίζεται σε reader quantity.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

080 / ΜΑΘΗΜΑ

Μικρός κατάλογος κρατήσεων · Εμπέδωση

Έννοια: Σύνθεση αντικειμένων, modules και συλλογών

Πριν αρχίσεις: 014, 015, 027, 076, 079

Χώρισε δεδομένα, επιλογή και εμφάνιση. Οι εγγραφές γνωρίζουν αν είναι ενεργές, ο κατάλογος επιλέγει και ο formatter ορίζει το κείμενο. Οι μικρές κλάσεις έχουν λίγες συγκεκριμένες ευθύνες.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class ActiveGuest
  attr_reader :name
  def initialize(name)
    @name = name
  end
  def active?
    true
  end
end
[ActiveGuest.new("Eva")].select { |g| g.active? }.map { |g| g.name }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε CatalogBooking.new(name, active) με readers name και active?. Γράψε BookingCatalog.new(bookings) και render(formatter): μόνο ενεργές κρατήσεις, formatter.format(name) για καθεμία, ένωση με newline χωρίς τελικό newline. Δώσε SimpleBookingFormat (ίδιο όνομα) και FramedBookingFormat ("[NAME]").

RUBY
```ruby
list = [CatalogBooking.new("Eva", true), CatalogBooking.new("Jo", false)]
BookingCatalog.new(list).render(FramedBookingFormat.new)
# => "[Eva]"
```

RUBY
```ruby
BookingCatalog.new([CatalogBooking.new("A", true), CatalogBooking.new("B",
true)]).render(SimpleBookingFormat.new)
# => "A\nB"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 080
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 080 --solution
```

Μια μικρή παραλλαγή

Φτιάξε formatter που προσθέτει "Guest: " και δοκίμασέ τον μόνο του.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Guest: Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

081 / ΜΑΘΗΜΑ

Απόρριψη άκυρης κλήσης

Έννοια: `raise` και `ArgumentError`

Πριν αρχίσεις: 009, 068

Η `raise ErrorClass, "message"` σταματά την κανονική ροή και δημιουργεί εξαίρεση. Η ArgumentError ταιριάζει σε μη αποδεκτή τιμή ορίσματος. Ο έλεγχος πρέπει να προηγείται του υπολογισμού που στηρίζεται στην τιμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def positive_fee(n)
  raise ArgumentError, "positive required" if n <= 0
  n * 2
end
positive_fee(3)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `checked_cost(seats, price)`. Τα seats και price είναι ακέραιοι. Αν seats <= 0 κάνε raise ArgumentError με μήνυμα "seats must be positive". Αν price < 0, "price must be nonnegative". Με αυτή τη σειρά ελέγχων. Αλλιώς δώσε γινόμενο.

```ruby
checked_cost(2, 4)
# => 8
```

```ruby
checked_cost(2, 0)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 081
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 081 --solution
```

Μια μικρή παραλλαγή

Φτιάξε έλεγχο μήκους ονόματος: άδειο String προκαλεί ArgumentError, άλλο String επιστρέφεται.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

082 / ΜΑΘΗΜΑ

Χειρισμός συγκεκριμένου σφάλματος

Έννοια: `rescue` με συγκεκριμένη κλάση

Πριν αρχίσεις: 081, 002

Η `Integer(text, 10)` κάνει αυστηρή μετατροπή δεκαδικού κειμένου και προκαλεί ArgumentError για άκυρη μορφή. Ένα συγκεκριμένο `rescue ArgumentError` δίνει ελεγχόμενη εναλλακτική. Το to_i είναι πιο επιεικές και δεν είναι κατάλληλο όταν πρέπει να απορρίπτεται το "3x".

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
begin
  Integer("x", 10)
rescue ArgumentError
  nil
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `parse_seats(text)`. Δέχεται String. Μετέτρεψέ το με Integer(text, 10). Αν αποτύχει με ArgumentError, επίστρεψε nil. Το 0 και οι αρνητικοί αριθμοί είναι έγκυρα αποτελέσματα μετατροπής.

RUBY
```ruby
parse_seats("12")
# => 12
```

RUBY
```ruby
parse_seats("3x")
# => nil
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 082
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 082 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε rescue για να δώσεις το String "invalid" όταν αποτύχει αυστηρή μετατροπή.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"invalid"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

083 / ΜΑΘΗΜΑ

Το σφάλμα ως αντικείμενο

Έννοια: `rescue => error`, `message` και `backtrace`

Πριν αρχίσεις: 082

Με `rescue ArgumentError => error` κρατάς το αντικείμενο εξαίρεσης. Η message δίνει το μήνυμα και η backtrace θέσεις κλήσεων. Το μήνυμα είναι String. Δεν χρειάζεται να εμφανίσεις ολόκληρο backtrace στον χρήστη μιας μικρής εφαρμογής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
begin
  raise ArgumentError, "missing title"
rescue ArgumentError => error
  error.message
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `validation_message(valid)`. Δέχεται boolean. Αν valid, επίστρεψε {ok: true, message: "ready"}. Αλλιώς κάνε raise ArgumentError, "invalid booking" και στο rescue επίστρεψε {ok: false, message: error.message}. Εδώ το τοπικό raise/rescue είναι άσκηση παρατήρησης της εξαίρεσης.

```ruby
validation_message(true)
# => {ok: true, message: "ready"}
```

```ruby
validation_message(false)
# => {ok: false, message: "invalid booking"}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 083
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 083 --solution
```

Μια μικρή παραλλαγή

Μετά από raise, έλεγξε ότι το πρώτο στοιχείο backtrace είναι String.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

084 / ΜΑΘΗΜΑ

Τιμή από επιτυχία ή αποτυχία

Έννοια: `begin` ως έκφραση και `else` σε χειρισμό σφαλμάτων

Πριν αρχίσεις: 082, 083

Το `begin ... rescue ... else ... end` είναι έκφραση. Το else εκτελείται μόνο όταν το προστατευμένο σώμα τελειώσει χωρίς εξαίρεση. Σφάλμα που δημιουργείται μέσα στο else δεν πιάνεται από την ίδια rescue. Κράτα στο begin μόνο την πράξη που περιμένεις ότι μπορεί να αποτύχει.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
begin
  value = Integer("4", 10)
rescue ArgumentError
  "bad"
else
  "value=#{value}"
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `conversion_label(text)`. Δέχεται String. Δοκίμασε Integer(text, 10) μέσα σε begin. Αν αποτύχει δώσε "invalid". Στο else δώσε "seats=N" με την ακέραιη τιμή.

```ruby
conversion_label("03")
# => "seats=3"
```

```ruby
conversion_label("three")
# => "invalid"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 084
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 084 --solution
```

Μια μικρή παραλλαγή

Σε επιτυχημένη μετατροπή "0" επίστρεψε Hash {value: 0}, σε αποτυχία nil.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{value: 0}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

085 / ΜΑΘΗΜΑ

Μικρός πίνακας αποτελεσμάτων · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 014, 082, 084

Κάθε γραμμή πρέπει να δώσει ρητό αποτέλεσμα επιτυχίας ή αποτυχίας. Η map κρατά μία απάντηση ανά είσοδο και το begin/rescue αποφασίζει τη μορφή κάθε απάντησης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["2", "x"].map do |text|
  begin
    Integer(text, 10)
  rescue ArgumentError
    nil
  end
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `conversion_results(texts)`. Για Array Strings δώσε ένα Hash ανά στοιχείο: {ok: true, value: N} όταν Integer(text, 10) πετυχαίνει, αλλιώς {ok: false, value: nil}. Κράτησε σειρά και μηδενικές τιμές.

```ruby
conversion_results(["2", "x", "0"])
# => [{ok: true, value: 2}, {ok: false, value: nil}, {ok: true, value: 0}]
```

```ruby
conversion_results([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 085
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 085 --solution
```

Μια μικρή παραλλαγή

Από έτοιμα conversion results μέτρησε τα ok.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `2`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

086 / ΜΑΘΗΜΑ

Ενέργεια που γίνεται πάντα

Έννοια: `ensure`

Πριν αρχίσεις: 081, 082, 076

Η `ensure` εκτελείται πριν φύγει ο έλεγχος από το προστατευμένο σώμα, είτε κανονικά είτε με εξαίρεση. Η τελική της έκφραση δεν αντικαθιστά από μόνη της το αποτέλεσμα. Μην βάζεις return στην ensure, γιατί μπορεί να κρύψει αποτέλεσμα ή εξαίρεση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
events = []
result = begin
  events << "work"
  7
ensure
  events << "cleanup"
end
[result, events]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `tracked_operation(events, fail_now)`. Το events είναι Array που επιτρέπεται να ενημερώσεις. Πρόσθεσε "start". Αν fail_now true, κάνε raise ArgumentError, "failed". Αλλιώς επίστρεψε "done". Σε κάθε περίπτωση πρόσθεσε "cleanup" ακριβώς μία φορά μέσω ensure. Μην καταπιείς την εξαίρεση.

```ruby
events = []
value = tracked_operation(events, false)
[value, events]
# => ["done", ["start", "cleanup"]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 086
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 086 --solution
```

Μια μικρή παραλλαγή

Παρατήρησε ensure μαζί με return από μέθοδο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[4, ["closed"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

087 / ΜΑΘΗΜΑ

Δικό σου είδος αποτυχίας

Έννοια: Custom exception κάτω από `StandardError`

Πριν αρχίσεις: 077, 081, 082

Μια δική σου εξαίρεση συνήθως κληρονομεί από StandardError. Έτσι έχει σαφές νόημα και μπορεί να πιαστεί συγκεκριμένα. Η κλάση ονομάζει την αποτυχία· το message περιγράφει το συγκεκριμένο συμβάν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class MissingTitle < StandardError
end
begin
  raise MissingTitle, "title required"
rescue MissingTitle => error
  error.message
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε InvalidQuantity < StandardError και validated_quantity(quantity). Η είσοδος είναι ακέραιος. Για αρνητικό κάνε raise InvalidQuantity, "negative quantity". Για 0 ή θετικό επίστρεψε την ίδια τιμή.

```ruby
validated_quantity(4)
# => 4
```

```ruby
validated_quantity(0)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 087
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 087 --solution
```

Μια μικρή παραλλαγή

Πιάσε συγκεκριμένη custom εξαίρεση και επίστρεψε false.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `false`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

088 / ΜΑΘΗΜΑ

Απουσία αποτελέσματος χωρίς εξαίρεση

Έννοια: `Integer(..., exception: false)` και ρητή πολιτική fallback

Πριν αρχίσεις: 048, 062, 082

Η `Integer(text, 10, exception: false)` δίνει nil αντί για εξαίρεση σε αποτυχία μετατροπής. Αυτό ταιριάζει με filter_map, που απορρίπτει nil αλλά διατηρεί το 0. Είναι επιλογή του συγκεκριμένου API, όχι γενικό keyword κάθε μεθόδου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["2", "x", "0"].filter_map { |s| Integer(s, 10, exception: false) }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `valid_integers(texts)`. Για Array Strings δώσε τις επιτυχημένες μετατροπές Integer βάσης 10. Απόρριψε άκυρες μορφές, κράτησε αρνητικά, μηδέν, σειρά και διπλότυπα. Χρησιμοποίησε exception: false και filter_map.

```ruby
valid_integers(["2", "x", "0", "-1", "2"])
# => [2, 0, -1, 2]
```

```ruby
valid_integers(["3x", ""])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 088
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 088 --solution
```

Μια μικρή παραλλαγή

Κράτησε μόνο μη αρνητικά από τις επιτυχημένες μετατροπές.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[0, 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

089 / ΜΑΘΗΜΑ

Μικρός εισαγωγέας εγγραφών · Εμπέδωση

Έννοια: Σύνθεση μετατροπών και συγκεκριμένων σφαλμάτων

Πριν αρχίσεις: 024, 026, 037, 062, 088

Η εισαγωγή χωρίζεται σε πεδία, αυστηρή μετατροπή και απλούς κανόνες αποδοχής. Όλα τα δεδομένα βρίσκονται ήδη στη μνήμη. Κράτησε κάθε βήμα ορατό για να ξεχωρίζει η σύνταξη από τη λογική.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
fields = "Eva:2".split(":", -1)
{name: fields[0], seats: Integer(fields[1], 10, exception: false)}
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `import_bookings(lines)`. Κάθε String περιέχει ακριβώς ένα : και δεν περιέχει άλλα πεδία. Καθάρισε όνομα, μετέτρεψε δεύτερο πεδίο αυστηρά σε ακέραιο. Κράτησε μόνο μη κενό όνομα και seats > 0. Επίστρεψε Hashes με name το καθαρό όνομα και seats την ποσότητα. Μην αλλάξεις lines.

RUBY
```ruby
import_bookings([" Eva :2", " :3", "Jo:x", "A:0", "B:-1", "C:"])
# => [{name: "Eva", seats: 2}]
```

RUBY
```ruby
import_bookings(["B:1", "A:2", "B:1"])
# => [{name: "B", seats: 1}, {name: "A", seats: 2}, {name: "B", seats: 1}]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 089
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 089 --solution
```

Μια μικρή παραλλαγή

Από ήδη έγκυρες εγγραφές υπολόγισε συνολικές θέσεις.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

090 / ΜΑΘΗΜΑ

Κώδικας σε περισσότερα αρχεία

Έννοια: `require`, `require_relative` και ρόλος του `load`

Πριν αρχίσεις: 074

Η `require` φορτώνει βιβλιοθήκη μία φορά και επιστρέφει true στην πρώτη φόρτωση, false αν είναι ήδη φορτωμένη. Η `require_relative` ψάχνει σχετικά με το αρχείο κώδικα. Η `load` εκτελεί ξανά ένα αρχείο και συνήθως θέλει την κατάληξη .rb. Το `__dir__` δίνει τον φάκελο του τρέχοντος αρχείου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
first = require_relative "support/demo"
second = require_relative "support/demo"
[first, second, LoadDemo.text]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `helper_greeting(name)`. Το support/greeting.rb παρέχει CourseGreeting.text(name). Φόρτωσέ το με require_relative και κάλεσέ το για να δώσεις "Hello NAME". Μην αντιγράψεις τον βοηθητικό κώδικα στην προσπάθεια.

```ruby
helper_greeting("Eva")
# => "Hello Eva"
```

```ruby
helper_greeting("Jo")
# => "Hello Jo"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 090
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 090 --solution
```

Μια μικρή παραλλαγή

Με load φόρτωσε το ίδιο demo αρχείο μέσω πλήρους τοπικού path και κάλεσε τη μέθοδό του.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"loaded"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

091 / ΜΑΘΗΜΑ

Βιβλιοθήκες και εξαρτήσεις

Έννοια: Core, standard/default/bundled gems, Gemfile και lockfile

Πριν αρχίσεις: 008, 090

Core κλάσεις όπως Array υπάρχουν χωρίς require. Η standard library περιλαμβάνει default και bundled gems που συνοδεύουν τη Ruby. Στη Ruby 3.4 το JSON είναι default gem και το CSV bundled. Το RSpec είναι πρόσθετο gem. Το Gemfile δηλώνει απαιτήσεις, το Gemfile.lock την επιλεγμένη επίλυση εκδόσεων. Με Bundler τα bundled gems που χρησιμοποιείς χρειάζονται δήλωση. Η παρούσα σειρά χρησιμοποιεί τις ήδη εγκατεστημένες βιβλιοθήκες και RSpec 3.13.2 μέσω bin/lesson, χωρίς εγκατάσταση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
Array.name
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `library_category(name)`. Με βάση τη Ruby 3.4 επίστρεψε "core" για "Array", "default" για "json", "bundled" για "csv", "external" για "rspec" και nil για άλλη τιμή. Έπειτα άνοιξε τα διπλανά Gemfile και Gemfile.lock και εντόπισε δήλωση και επιλεγμένη έκδοση RSpec, χωρίς να τα αλλάξεις.

```ruby
library_category("Array")
# => "core"
```

```ruby
library_category("csv")
# => "bundled"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 091
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 091 --solution
```

Μια μικρή παραλλαγή

Φόρτωσε JSON και επιβεβαίωσε ότι το όνομα του module είναι "JSON".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"JSON"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

092 / ΜΑΘΗΜΑ

Γράψε δικό σου παράδειγμα συμπεριφοράς

Έννοια: RSpec `describe`, `it`, `expect`

Πριν αρχίσεις: 013, 081, 090

Η `RSpec.describe` ομαδοποιεί tests. Κάθε `it` ονομάζει μία αναμενόμενη συμπεριφορά. Το `expect(actual).to eq(expected)` συγκρίνει αποτέλεσμα με την ανεξάρτητα γραμμένη απαίτηση. Σε αυτό το μάθημα συμπληρώνεις tests για έτοιμη μέθοδο, όχι την ίδια τη μέθοδο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "rspec/expectations"
include RSpec::Matchers
expect(2 + 3).to eq(5)
true
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Το support/cost.rb δίνει ticket_total(seats, price). Γράψε στο exercise.rb τρία it: ticket_total(3, 4) == 12, ticket_total(0, 4) == 0 και ticket_total(2, 0) == 0. Δώσε περιγραφές που ξεχωρίζουν κανονική κράτηση, μηδενικές θέσεις και δωρεάν είσοδο.

```ruby
ticket_total(3, 4)
# => 12
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 092
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 092 --solution
```

Μια μικρή παραλλαγή

Γράψε έναν έλεγχο ότι ένα κενό άθροισμα επιστρέφει 0.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

093 / ΜΑΘΗΜΑ

Έλεγχος αποτυχίας και μεταβολής

Έννοια: RSpec για exceptions, ταυτότητα και είσοδο

Πριν αρχίσεις: 035, 081, 092

Με `expect { call }.to raise_error(Class, message)` ελέγχεις εξαίρεση: το block καθυστερεί την κλήση. Η eq συγκρίνει τιμές, ενώ η be(object) ταυτότητα. Για μη μεταβολή κράτα ξεχωριστή αναμενόμενη τιμή και έλεγξε την είσοδο μετά την κλήση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
require "rspec/expectations"
include RSpec::Matchers
a = [1]
b = a.dup
expect(b).to eq(a)
expect(b).not_to be(a)
true
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Το support/names.rb παρέχει upper_names(names), που επιστρέφει νέο Array κεφαλαίων Strings και κάνει raise ArgumentError, "names required" για nil. Γράψε tests για αποτέλεσμα, μη μεταβολή εισόδου και των εσωτερικών Strings, νέα ταυτότητα Array, κενό Array και το error.

RUBY
```ruby
upper_names(["Eva"])
# => ["EVA"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 093
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 093 --solution
```

Μια μικρή παραλλαγή

Έλεγξε ότι η raise μεταδίδει και τη σωστή κλάση και το μήνυμα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

094 / ΜΑΘΗΜΑ

Ανάγνωση ολόκληρου κειμένου

Έννοια: `File.read` και ρητό encoding

Πριν αρχίσεις: 024, 032, 048, 090

Η `File.read(path, encoding: "UTF-8")` ανοίγει, διαβάζει όλο το μικρό αρχείο και το κλείνει. Επιστρέφει String. Η ρητή κωδικοποίηση δηλώνει πώς ερμηνεύεται το κείμενο. Τα paths της άσκησης δείχνουν μόνο στα δοσμένα fixtures.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
File.read(__dir__ + "/fixtures/names.txt", encoding: "UTF-8").lines.length
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `read_names(path)`. Διάβασε υπάρχον μικρό UTF-8 αρχείο, χώρισε γραμμές, καθάρισε άκρα και αφαίρεσε κενές. Επίστρεψε Array ονομάτων με αρχική σειρά. Άφησε σφάλματα πρόσβασης να διαδοθούν.

```ruby
read_names(__dir__ + "/fixtures/names.txt")
# => ["Εύα", "Άρης"]
```

```ruby
read_names(__dir__ + "/fixtures/empty.txt")
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 094
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 094 --solution
```

Μια μικρή παραλλαγή

Διάβασε το ίδιο fixture και κράτησε τις γραμμές με chomp, χωρίς αφαίρεση κενών.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[" Εύα ", "", "Άρης"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

095 / ΜΑΘΗΜΑ

Άνοιγμα με ορισμένο χρόνο ζωής

Έννοια: `File.open` με block και modes

Πριν αρχίσεις: 086, 094

Η `File.open(path, "r:UTF-8") { |file| ... }` δίνει ανοικτό File στο block και το κλείνει αυτόματα όταν φύγεις, ακόμη και με εξαίρεση. Επιστρέφει την τιμή του block. Το mode r είναι ανάγνωση, w αντικατάσταση περιεχομένου και a προσθήκη στο τέλος.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
File.open(__dir__ + "/fixtures/message.txt", "r:UTF-8") { |file| file.read.chomp }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `read_and_track(path, audit)`. Άνοιξε δοσμένο αρχείο μόνο για ανάγνωση UTF-8 με block. Πρόσθεσε το αντικείμενο File στο audit Array, διάβασε όλο το περιεχόμενο και επίστρεψέ το. Μετά την επιστροφή το καταγεγραμμένο File πρέπει να είναι κλειστό. Μόνο το audit αλλάζει.

```ruby
audit = []
text = read_and_track(__dir__ + "/fixtures/message.txt", audit)
[text, audit.length, audit.first.closed?]
# => ["Ready\n", 1, true]
```

```ruby
audit = ["before"]
read_and_track(__dir__ + "/fixtures/message.txt", audit)
[audit.first, audit.length]
# => ["before", 2]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 095
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 095 --solution
```

Μια μικρή παραλλαγή

Κάνε raise μέσα σε File.open block και επιβεβαίωσε ότι το File έκλεισε.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

096 / ΜΑΘΗΜΑ

Γραμμές μία μία

Έννοια: `File.foreach` με block

Πριν αρχίσεις: 013, 030, 032, 094

Η `File.foreach(path, encoding: "UTF-8")` με block δίνει μία γραμμή τη φορά. Το αρχείο κλείνει όταν τελειώσει η διάσχιση. Το newline παραμένει στη γραμμή. Εδώ δίνουμε πάντα block· τη μορφή χωρίς block θα τη δεις στους Enumerators.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
count = 0
File.foreach(__dir__ + "/fixtures/log.txt", encoding: "UTF-8") { |line| count =
count + 1 }
count
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `error_lines(path)`. Διάβασε δοσμένο UTF-8 αρχείο γραμμή γραμμή. Κράτησε μόνο γραμμές που αρχίζουν με "ERROR:" και αφαίρεσε το τελικό newline με chomp. Επίστρεψε Array, χωρίς αλλαγή αρχείου.

```ruby
error_lines(__dir__ + "/fixtures/log.txt")
# => ["ERROR: missing"]
```

```ruby
error_lines(__dir__ + "/fixtures/empty.txt")
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 096
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 096 --solution
```

Μια μικρή παραλλαγή

Με foreach μέτρησε γραμμές που αρχίζουν INFO:.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `1`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

097 / ΜΑΘΗΜΑ

Παραγωγή αρχείου

Έννοια: `File.write`, append και διαδρομή εξόδου

Πριν αρχίσεις: 027, 095

Η `File.write(path, text, mode: "w:UTF-8")` αντικαθιστά το περιεχόμενο και επιστρέφει bytes που γράφτηκαν. Το mode a προσθέτει στο τέλος. Για την άσκηση χρησιμοποιείς μόνο αρχεία μέσα στο tmp του μαθήματος, που προορίζονται για έξοδο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
path = __dir__ + "/tmp/example.txt"
File.write(path, "A\n", mode: "w:UTF-8")
File.write(path, "B\n", mode: "a:UTF-8")
File.read(path, encoding: "UTF-8")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `save_lines(path, lines, append: false)`. Το path δίνεται για τοπική έξοδο. Γράψε κάθε String του lines με τελικό newline, με αντικατάσταση ή append ανά keyword. Κενό lines γράφει κενό κείμενο. Επίστρεψε το path, όχι byte count. Μην αλλάξεις lines.

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 097
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 097 --solution
```

Μια μικρή παραλλαγή

Γράψε μία ελληνική γραμμή σε tmp/variation.txt και διάβασέ την πίσω.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Γεια\n"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

098 / ΜΑΘΗΜΑ

Διαδρομές και επιλογή αρχείων

Έννοια: `File.join`, `Dir.glob` και `Pathname`

Πριν αρχίσεις: 053, 090, 094

Η `File.join` συνθέτει μέρη διαδρομής. Η `Dir.glob` βρίσκει ονόματα που ταιριάζουν σε μοτίβο όπως *.txt. Η Pathname είναι αντικείμενο διαδρομής με μεθόδους όπως basename και extname· η to_s δίνει String. Η δημιουργία Pathname δεν διαβάζει το αρχείο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "pathname"
path = Pathname.new("reports/day.csv")
[path.basename.to_s, path.extname]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `text_filenames(directory)`. Στον δοσμένο φάκελο fixtures υπάρχουν μόνο κανονικά αρχεία. Βρες τα *.txt του ίδιου επιπέδου με Dir.glob και File.join. Επίστρεψε μόνο τα basenames ως ταξινομημένα Strings, χωρίς ανάγνωση περιεχομένων.

```ruby
text_filenames(__dir__ + "/fixtures")
# => ["a.txt", "b.txt"]
```

```ruby
text_filenames(__dir__ + "/fixtures/empty")
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 098
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 098 --solution
```

Μια μικρή παραλλαγή

Σύνθεσε path από τρία μέρη χωρίς χειροκίνητα slash.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"reports/2026/day.csv"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

099 / ΜΑΘΗΜΑ

Ορίσματα από το Terminal

Έννοια: `ARGV`

Πριν αρχίσεις: 011, 038, 090

Το `ARGV` είναι Array Strings από τα ορίσματα μετά το όνομα του Ruby αρχείου. Το όνομα του script δεν περιλαμβάνεται. Με παράμετρο `args = ARGV` μπορείς να δοκιμάζεις τη λογική με δικό σου Array. Οι θέσεις που λείπουν δίνουν nil.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
args = ["names.txt", "report.txt"]
{input: args[0], output: args[1]}
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `command_paths(args = ARGV)`. Αν args έχει ακριβώς δύο Strings, δώσε {input: args[0], output: args[1]}. Αλλιώς κάνε raise ArgumentError, "expected input and output". Μην αλλάξεις args. Δίνεται cli.rb για κλήση `ruby cli.rb names.txt report.txt` μετά τη λύση.

```ruby
command_paths(["names.txt", "report.txt"])
# => {input: "names.txt", output: "report.txt"}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 099
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 099 --solution
```

Μια μικρή παραλλαγή

Από δοσμένο Array args διάβασε προαιρετικό τρίτο όρισμα με default "plain".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"plain"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

100 / ΜΑΘΗΜΑ

Ροές εισόδου και εξόδου

Έννοια: `STDIN`, `STDOUT`, `STDERR` και `StringIO`

Πριν αρχίσεις: 076, 095

Τα STDIN, STDOUT και STDERR είναι αντικείμενα ροών: είσοδος, κανονική έξοδος και σφάλματα. Οι gets και puts διαβάζουν/γράφουν γραμμές. Με `require "stringio"` φτιάχνεις ροή μνήμης, με ίδιο μικρό API, για tests χωρίς πληκτρολόγηση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
require "stringio"
input = StringIO.new("Eva\n")
input.gets.chomp
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `greet_stream(input = STDIN, output = STDOUT, errors = STDERR)`. Διάβασε μία γραμμή με gets. Αν δεν υπάρχει ή είναι κενή μετά από strip, γράψε "name required" στο errors και δώσε false. Αλλιώς γράψε "Hello NAME" στο output και δώσε true. Κάθε μήνυμα μία γραμμή. Μην κλείσεις τις ροές του καλούντος.

RUBY
```ruby
require "stringio"
out = StringIO.new
err = StringIO.new
ok = greet_stream(StringIO.new(" Eva \n"), out, err)
[ok, out.string, err.string]
# => [true, "Hello Eva\n", ""]
```

RUBY
```ruby
require "stringio"
out = StringIO.new
err = StringIO.new
ok = greet_stream(StringIO.new(""), out, err)
[ok, out.string, err.string]
# => [false, "", "name required\n"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 100
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 100 --solution
```

Μια μικρή παραλλαγή

Γράψε δύο γραμμές σε StringIO και δώσε το τελικό String.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"A\nB\n"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

101 / ΜΑΘΗΜΑ

JSON

Έννοια: `JSON.parse` και `JSON.generate`

Πριν αρχίσεις: 037, 090

Με `require "json"`, η JSON.parse μετατρέπει JSON String σε Ruby Arrays, Hashes και απλές τιμές. Τα κλειδιά είναι Strings από προεπιλογή. Η JSON.generate κάνει την αντίστροφη διαδρομή. Το JSON null γίνεται nil και το false παραμένει false.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
require "json"
JSON.parse('{"name":"Eva","active":false,"note":null}')
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `json_names(text)`. Το έγκυρο JSON περιέχει Array objects με String πεδίο "name" και boolean "active". Επίστρεψε JSON String με Array ονομάτων μόνο των ενεργών εγγραφών, με αρχική σειρά. Δεν χρειάζεται χειρισμός άκυρου JSON.

RUBY
```ruby
json_names('[{"name":"Eva","active":true},{"name":"Jo","active":false}]')
# => '["Eva"]'
```

RUBY
```ruby
json_names("[]")
# => "[]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 101
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 101 --solution
```

Μια μικρή παραλλαγή

Κάνε generate και parse ενός Hash με false και nil και δες τα String κλειδιά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{"active" => false, "note" => nil}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

102 / ΜΑΘΗΜΑ

CSV

Έννοια: `CSV` με headers

Πριν αρχίσεις: 026, 088, 090

Η CSV.parse με `headers: true` διαβάζει ονόματα στηλών από την πρώτη γραμμή. Κάθε row δίνει πρόσβαση με String όνομα. Η CSV χειρίζεται quotes, κόμματα και αλλαγές γραμμής μέσα σε πεδίο· ένα απλό split(",") δεν τα καλύπτει.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "csv"
CSV.parse("name,seats\n\"Eva, Jr\",2\n", headers: true).first["name"]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `csv_bookings(text)`. Δέχεται έγκυρο CSV με headers name,seats, έγκυρα μη κενά ονόματα και δεκαδικούς ακεραίους seats. Δώσε Array Hashes {name: String, seats: Integer}, με αρχική σειρά. Χρησιμοποίησε CSV.parse, όχι split.

```ruby
csv_bookings("name,seats\n\"Eva, Jr\",2\nJo,0\n")
# => [{name: "Eva, Jr", seats: 2}, {name: "Jo", seats: 0}]
```

```ruby
csv_bookings("name,seats\n")
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 102
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 102 --solution
```

Μια μικρή παραλλαγή

Διάβασε όνομα που περιέχει escaped διπλό εισαγωγικό από CSV.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `'Eva "E"'`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

103 / ΜΑΘΗΜΑ

Ρυθμίσεις από το περιβάλλον

Έννοια: `ENV` και ρητές προεπιλογές

Πριν αρχίσεις: 038, 088, 090

Το ENV παρέχει μεταβλητές περιβάλλοντος ως Strings ή nil. Η fetch δίνει default μόνο όταν λείπει όνομα. Χρησιμοποιούμε αποκλειστικά επινοημένα ονόματα RUBY_LESSON_* και δοσμένο Hash στα tests, χωρίς ανάγνωση άλλων ρυθμίσεων.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
demo_env = {"RUBY_LESSON_LIMIT" => "0"}
Integer(demo_env.fetch("RUBY_LESSON_LIMIT", "10"), 10)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `demo_settings(env = ENV)`. Διάβασε RUBY_LESSON_LABEL με default "Guest" και RUBY_LESSON_LIMIT με default "10". Το limit μετατρέπεται με Integer βάσης 10. Επίστρεψε {label: String, limit: Integer}. Κενό label και μηδενικό limit διατηρούνται. Άκυρο limit αφήνει ArgumentError να διαδοθεί.

```ruby
demo_settings({})
# => {label: "Guest", limit: 10}
```

```ruby
demo_settings({"RUBY_LESSON_LABEL" => "", "RUBY_LESSON_LIMIT" => "0"})
# => {label: "", limit: 0}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 103
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 103 --solution
```

Μια μικρή παραλλαγή

Από δοσμένο Hash διάβασε επινοημένο flag: ενεργό μόνο όταν η τιμή είναι ακριβώς "1".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `false`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

104 / ΜΑΘΗΜΑ

Αναφορά από fixtures · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 067, 094, 097

Η μικρή αναφορά παίρνει είσοδο από ένα δοσμένο αρχείο και γράφει σε δοσμένη έξοδο. Η λογική παραμένει η γνωστή αλυσίδα καθαρίσματος και ομαδοποίησης. Στο tmp του μαθήματος βρίσκονται μόνο αναλώσιμες έξοδοι.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
File.read(__dir__ + "/fixtures/names.txt", encoding: "UTF-8").lines.map { |s|
s.strip }.reject { |s| s.empty? }.tally
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `write_name_counts(input_path, output_path)`. Διάβασε μικρό UTF-8 αρχείο ASCII ονομάτων, ένα ανά γραμμή. Καθάρισε άκρα, απόρριψε κενά, μέτρησε όμοιες τιμές και γράψε αύξοντα `NAME: COUNT`, με newline μετά από κάθε γραμμή. Κενή είσοδος γράφει κενό αρχείο. Επίστρεψε πλήθος διαφορετικών ονομάτων.

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 104
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 104 --solution
```

Μια μικρή παραλλαγή

Από έτοιμα πλήθη δώσε το συνολικό πλήθος συμμετοχών.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `3`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

105 / ΜΑΘΗΜΑ

Τοπικό εργαλείο αναφορών · Εμπέδωση

Έννοια: Σύνθεση αρχείων, parsing, συλλογών και tests

Πριν αρχίσεις: 088, 094, 097, 101, 102

Σύνδεσε CSV είσοδο με JSON έξοδο και κράτησε τη μετατροπή σε μικρά βήματα. Τα paths είναι ρητά ορίσματα. Η αναφορά περιλαμβάνει μόνο αποδεκτές εγγραφές, χωρίς νέα αλγοριθμική τεχνική.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "csv"
require "json"
rows = CSV.parse("name,seats\nEva,2\n", headers: true)
JSON.generate(rows.map { |row| {name: row["name"], seats: Integer(row["seats"], 10)}
})
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `convert_booking_file(input_path, output_path)`. Το input είναι έγκυρο UTF-8 CSV με headers name,seats. Καθάρισε name, απόρριψε κενό ή απόν name και seats που δεν μετατρέπεται σε θετικό δεκαδικό ακέραιο. Γράψε JSON Array {name, seats} στο δοσμένο τοπικό output. Επίστρεψε πλήθος αποδεκτών. Μην αλλάξεις το input.

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 105
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 105 --solution
```

Μια μικρή παραλλαγή

Από έτοιμο JSON Array εγγραφών πάρε συνολικές θέσεις.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

106 / ΜΑΘΗΜΑ

Η δική σου μέθοδος δέχεται block

Έννοια: `yield`

Πριν αρχίσεις: 013, 014, 004

Η `yield(value)` εκτελεί το block που δόθηκε στην τρέχουσα μέθοδο. Η παράμετρος του block παίρνει το value και η επιστροφή του γίνεται η τιμή της yield. Χωρίς block η yield προκαλεί LocalJumpError. Έτσι μια δική σου μέθοδος κρατά τη ροή και ο καλών επιλέγει τη μορφοποίηση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def surround(value)
  "[#{yield(value)}]"
end
surround("Eva") { |name| name.upcase }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `format_lines(lines)`. Για Array Strings κάλεσε το δοσμένο block μία φορά για κάθε γραμμή και επίστρεψε Array των αποτελεσμάτων. Η χρήση με μη κενή είσοδο απαιτεί block. Μην αλλάξεις lines.

RUBY
```ruby
format_lines(["a", "b"]) { |s| s.upcase }
# => ["A", "B"]
```

RUBY
```ruby
seen = []
result = format_lines(["A", "B"]) { |s| seen << s; "[#{s}]" }
[result, seen]
# => [["[A]", "[B]"], ["A", "B"]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 106
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 106 --solution
```

Μια μικρή παραλλαγή

Δώσε δύο τιμές στο ίδιο block και πάρε το άθροισμά τους.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

107 / ΜΑΘΗΜΑ

Το block είναι προαιρετικό

Έννοια: `block_given?`

Πριν αρχίσεις: 106

Η `block_given?` απαντά αν η τρέχουσα μέθοδος έλαβε block. Μπορείς να δώσεις default συμπεριφορά χωρίς block και να επιστρέψεις ακριβώς το αποτέλεσμα του block όταν υπάρχει, ακόμη κι αν αυτό είναι nil ή false.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def greet_optional(name)
  return yield(name) if block_given?
  "Hi #{name}"
end
greet_optional("Eva")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `flexible_label(name)`. Χωρίς block επίστρεψε "Guest NAME". Με block, πέρασε name και επίστρεψε την ακριβή τιμή του block. Μην αντικαταστήσεις false ή nil με default.

```ruby
flexible_label("Eva")
# => "Guest Eva"
```

```ruby
flexible_label("Eva") { |s| s.upcase }
# => "EVA"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 107
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 107 --solution
```

Μια μικρή παραλλαγή

Κάλεσε προαιρετικό block δύο φορές όταν υπάρχει, αλλιώς δώσε [].

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[4, 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

108 / ΜΑΘΗΜΑ

Block ως τιμή

Έννοια: `Proc`, `proc`, `&block` και `call`

Πριν αρχίσεις: 106, 107

Το `proc { ... }` δημιουργεί αντικείμενο κώδικα. Με call το εκτελείς αργότερα. Το `&block` στην υπογραφή κρατά το δοσμένο block ως Proc ή nil αν δεν δόθηκε. Μπορείς να αποθηκεύεις callable αντικείμενα σε Array ή Hash όπως άλλες τιμές.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
formatter = proc { |name| "[#{name}]" }
formatter.call("Jo")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε capture_formatter(&block) που επιστρέφει το block ως Proc και apply_formatter(name, formatters, index) που επιλέγει το στοιχείο Array στη δοσμένη έγκυρη θέση και καλεί call(name).

```ruby
f = capture_formatter { |name| name.upcase }
f.call("Eva")
# => "EVA"
```

```ruby
formats = [proc { |s| s.upcase }, proc { |s| "[#{s}]" }]
apply_formatter("Eva", formats, 1)
# => "[Eva]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 108
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 108 --solution
```

Μια μικρή παραλλαγή

Αποθήκευσε proc για διπλασιασμό και κάλεσέ το με 4.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `8`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

109 / ΜΑΘΗΜΑ

Μεταβλητές που θυμάται ένα block

Έννοια: Closures

Πριν αρχίσεις: 108

Ένα Proc θυμάται το περιβάλλον στο οποίο δημιουργήθηκε. Αυτό λέγεται closure. Μπορεί να χρησιμοποιεί τοπικές μεταβλητές και αφού η μέθοδος κατασκευής έχει επιστρέψει. Οι μεταβλητές δεν γίνονται αυτόματα παγωμένα snapshots.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
prefix = "Hi"
f = proc { |name| "#{prefix} #{name}" }
prefix = "Hello"
f.call("Eva")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `prefix_formatter(prefix)`. Επίστρεψε Proc που δέχεται name και δίνει "PREFIX NAME". Κάθε κλήση της factory κρατά το δικό της prefix. Τα δοσμένα prefix Strings δεν μεταβάλλονται από τον καλούντα σε αυτή την άσκηση.

RUBY
```ruby
prefix_formatter("Hi").call("Eva")
# => "Hi Eva"
```

RUBY
```ruby
a = prefix_formatter("A")
b = prefix_formatter("B")
[a.call("Jo"), b.call("Jo")]
# => ["A Jo", "B Jo"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 109
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 109 --solution
```

Μια μικρή παραλλαγή

Φτιάξε closure που αυξάνει ιδιωτικό τοπικό μετρητή σε κάθε call.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[1, 2]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

110 / ΜΑΘΗΜΑ

Συνάρτηση με δικά της όρια

Έννοια: `lambda`, `->` και arity

Πριν αρχίσεις: 108, 109

Η lambda γράφεται `lambda { |x| ... }` ή `->(x) { ... }`. Είναι Proc με αυστηρότερο πλήθος ορισμάτων. Μια lambda με μία παράμετρο απορρίπτει κλήση με μηδέν ή δύο. Ένα συνηθισμένο proc είναι πιο επιεικές στην αντιστοίχιση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
double = ->(n) { n * 2 }
[double.call(3), double.arity]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `price_calculator(price)`. Επίστρεψε lambda μίας παραμέτρου quantity που δίνει quantity * price. Οι αριθμοί είναι μη αρνητικοί ακέραιοι. Η callable πρέπει να απορρίπτει λάθος πλήθος ορισμάτων με ArgumentError.

RUBY
```ruby
price_calculator(4).call(3)
# => 12
```

RUBY
```ruby
price_calculator(4).call(0)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 110
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 110 --solution
```

Μια μικρή παραλλαγή

Γράψε την ίδια ιδέα με τη λέξη lambda και πρόσθεση αντί για γινόμενο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `7`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

111 / ΜΑΘΗΜΑ

Από πού επιστρέφει το return;

Έννοια: Διαφορά `return` σε μέθοδο, Proc και lambda

Πριν αρχίσεις: 009, 108, 110

Το return μέσα σε lambda επιστρέφει από τη lambda. Σε συνηθισμένο proc επιχειρεί να επιστρέψει από τη μέθοδο μέσα στην οποία ορίστηκε. Αν εκείνη έχει ήδη τελειώσει, προκαλεί LocalJumpError. Χρησιμοποίησε τελική έκφραση όταν δεν χρειάζεσαι ειδική έξοδο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def lambda_boundary
  action = -> { return "inside" }
  [action.call, "after"]
end
lambda_boundary
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε lambda_flow: lambda με return "inside", μετά επίστρεψε [τιμή lambda, "after"]. Γράψε proc_flow: proc με return "inside", κάλεσέ το και έπειτα γράψε "after". Η δεύτερη μέθοδος πρέπει να επιστρέφει "inside".

RUBY
```ruby
lambda_flow
# => ["inside", "after"]
```

RUBY
```ruby
proc_flow
# => "inside"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 111
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 111 --solution
```

Μια μικρή παραλλαγή

Επίστρεψε proc με return από factory και πιάσε το LocalJumpError όταν το καλέσεις μετά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"expired return target"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

112 / ΜΑΘΗΜΑ

Callable με προσαρμογή · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 109, 110, 111

Μια factory μπορεί να αποθηκεύσει επιλογές σε closure και να επιστρέψει lambda με καθαρή υπογραφή. Δοκίμασε δύο αποτελέσματα της factory, ώστε να φανεί ότι το καθένα χρησιμοποιεί τις δικές του επιλογές.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def suffixer(suffix)
  ->(text) { "#{text}#{suffix}" }
end
suffixer("!").call("Hi")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `custom_formatter(prefix, uppercase: false)`. Επίστρεψε lambda που δέχεται ένα String name. Αν uppercase true, κάνε κεφαλαίο μόνο το name. Δώσε "PREFIX NAME". Prefix και name δεν μεταβάλλονται.

RUBY
```ruby
custom_formatter("Hi", uppercase: true).call("Eva")
# => "Hi EVA"
```

RUBY
```ruby
a = custom_formatter("A")
b = custom_formatter("B", uppercase: true)
[a.call("Jo"), b.call("Jo")]
# => ["A Jo", "B JO"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 112
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 112 --solution
```

Μια μικρή παραλλαγή

Φτιάξε lambda που θυμάται άνοιγμα και κλείσιμο ετικέτας.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"[Eva]"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

113 / ΜΑΘΗΜΑ

Υπάρχουσα μέθοδος ως callable

Έννοια: `method(:name)` και `Method#call`

Πριν αρχίσεις: 068, 108

Η `receiver.method(:name)` επιστρέφει Method δεμένο με αυτόν τον receiver. Με call εκτελείς την ήδη υπάρχουσα μέθοδο. Δεν χρειάζεται να ξαναγράψεις τη συμπεριφορά σε proc. Το όνομα δίνεται ως Symbol.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
formatter = "Eva".method(:upcase)
formatter.call
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε TextFormats με upper(name) και bracket(name). Η selected(style) επιστρέφει Method για upper όταν style == :upper και για bracket όταν style == :bracket. Άλλη τιμή προκαλεί ArgumentError, "unknown style".

```ruby
TextFormats.new.selected(:upper).call("Eva")
# => "EVA"
```

```ruby
TextFormats.new.selected(:bracket).call("Jo")
# => "[Jo]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 113
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 113 --solution
```

Μια μικρή παραλλαγή

Κράτησε Method του length δεμένο στο String "Ruby" και κάλεσέ το.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `4`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

114 / ΜΑΘΗΜΑ

Μετατροπή σε block

Έννοια: `&`, `to_proc` και `&:method_name`

Πριν αρχίσεις: 108, 113, 014

Στην κλήση, το `&callable` μετατρέπει ένα αντικείμενο σε block μέσω to_proc όταν χρειάζεται. Το Symbol#to_proc κάνει κλήση της ονομασμένης μεθόδου στο στοιχείο. Έτσι `map(&:upcase)` αντιστοιχεί εδώ σε `map { |name| name.upcase }`. Δεν εκφράζει κάθε σύνθετο block.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["Eva", "Jo"].map(&:upcase)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `shorthand_names(names)`. Για Array Strings καθάρισε άκρα και κάνε κεφαλαία με δύο map και Symbol shorthand. Κράτησε και κενά αποτελέσματα. Μην αλλάξεις την είσοδο.

```ruby
shorthand_names([" Eva ", "jo"])
# => ["EVA", "JO"]
```

```ruby
shorthand_names([" "])
# => [""]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 114
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 114 --solution
```

Μια μικρή παραλλαγή

Πέρασε αποθηκευμένη lambda ως block της map.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["[Eva]", "[Jo]"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

115 / ΜΑΘΗΜΑ

Μια αλυσίδα που εξηγείται · Εμπέδωση

Έννοια: Σύνθεση μετατροπών, `tap`, `then` και callable

Πριν αρχίσεις: 062, 064, 065, 108, 114

Δώσε όνομα στο αποτέλεσμα κάθε σταδίου: καθαρή λίστα, καταγραφή πλήθους, τελικό μήνυμα. Η tap παρατηρεί και η then μετατρέπει σε ό,τι επιστρέψει ο formatter.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[" A ", " "].map(&:strip).reject(&:empty?).then { |names| names.join(", ") }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `formatted_report(names, audit, formatter)`. Καθάρισε άκρα, απόρριψε κενά και πρόσθεσε το πλήθος καθαρών ονομάτων στο audit με tap. Με then κάλεσε formatter.call(cleaned_names) και επίστρεψε ακριβώς το αποτέλεσμά του. Μόνο audit επιτρέπεται να αλλάξει. Ο formatter δίνεται χωρίς μεταβολή εισόδου.

```ruby
audit = []
value = formatted_report([" A ", " "], audit, ->(names) { names.join(", ") })
[value, audit]
# => ["A", [1]]
```

```ruby
audit = []
value = formatted_report([], audit, ->(names) { false })
[value, audit]
# => [false, [0]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 115
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 115 --solution
```

Μια μικρή παραλλαγή

Με tap κράτησε την έτοιμη λίστα σε audit και με then δώσε το μήκος της.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, [["A", "B"]]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

116 / ΜΑΘΗΜΑ

Μεταβλητό πλήθος ορισμάτων

Έννοια: `*args` στον ορισμό

Πριν αρχίσεις: 004, 027

Το `*lines` στην υπογραφή μαζεύει όσα positional ορίσματα απομένουν σε Array. Με καμία τιμή δίνει []. Μπορείς να έχεις ένα υποχρεωτικό όρισμα πριν από το rest. Δεν χρειάζεται ο καλών να κατασκευάσει Array.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def joined_names(*names)
  names.join(", ")
end
joined_names("Eva", "Jo")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `message_lines(title, *lines)`. Title και lines είναι Strings. Επίστρεψε title και όλες τις γραμμές ενωμένες με newline, χωρίς τελικό newline. Με μόνο title δώσε το ίδιο κείμενο. Κράτησε κενές γραμμές.

```ruby
message_lines("Report", "A", "B")
# => "Report\nA\nB"
```

```ruby
message_lines("Report")
# => "Report"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 116
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 116 --solution
```

Μια μικρή παραλλαγή

Φτιάξε μέθοδο που δέχεται όσα ποσά θέλεις και τα αθροίζει.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `9`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

117 / ΜΑΘΗΜΑ

Άνοιγμα λίστας στην κλήση

Έννοια: Splat `*` στην κλήση και σε Array literal

Πριν αρχίσεις: 116

Το `*values` στην κλήση ανοίγει Array σε χωριστά positional ορίσματα. Μέσα σε Array literal, `[first, *others]` ανοίγει τα others ως στοιχεία. Είναι η αντίστροφη κατεύθυνση από το *args στον ορισμό.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = [2, 3]
[1, *values, 4]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Η pair_label(left, right) δίνεται έτοιμη και επιστρέφει "LEFT/RIGHT". Γράψε expanded_label(pair) που δέχεται Array ακριβώς δύο Strings και καλεί pair_label με splat. Μην χρησιμοποιήσεις χειροκίνητους δείκτες.

```ruby
expanded_label(["A", "B"])
# => "A/B"
```

```ruby
expanded_label(["", "B"])
# => "/B"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 117
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 117 --solution
```

Μια μικρή παραλλαγή

Πρόσθεσε αρχική και τελική ετικέτα γύρω από Array ονομάτων με splat literal.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["start", "Eva", "Jo", "end"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

118 / ΜΑΘΗΜΑ

Ονομασμένες επιλογές ως hash

Έννοια: `**kwargs`, `**options` και διάκριση Hash από keywords

Πριν αρχίσεις: 048, 116, 117

Το `**options` στον ορισμό συγκεντρώνει keywords σε Hash. Στην κλήση ανοίγει Hash ως keywords. Στη Ruby 3 ένα απλό positional Hash δεν μετατρέπεται αυτόματα σε keywords μιας μεθόδου. Χρησιμοποίησε ρητά ** όταν αυτή είναι η πρόθεση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def box_text(text, left: "[", right: "]")
  "#{left}#{text}#{right}"
end
options = {left: "<", right: ">"}
box_text("A", **options)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται heading(name, prefix: "", suffix: "!"). Γράψε forwarded_heading(name, **options) που προωθεί ακριβώς τα options στη heading. Άγνωστα keywords αφήνουν το ArgumentError της heading να διαδοθεί.

```ruby
forwarded_heading("Eva")
# => "Eva!"
```

```ruby
forwarded_heading("Eva", prefix: "Hi ", suffix: "?")
# => "Hi Eva?"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 118
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 118 --solution
```

Μια μικρή παραλλαγή

Συγκέντρωσε keywords σε Hash και επίστρεψέ τα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{color: "blue", size: 2}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

119 / ΜΑΘΗΜΑ

Η ίδια μέθοδος με διαφορετικές κλήσεις · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 047, 048, 117, 118

Πριν καλέσεις μια μέθοδο, αντιστοίχισε τις positional τιμές βάσει θέσης και τα keywords βάσει ονόματος. Τα * και ** κάνουν διαφορετική δουλειά και μπορούν να εμφανίζονται στην ίδια κλήση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def count_label(name, quantity = 1, prefix: "")
  "#{prefix}#{name}: #{quantity}"
end
count_label(*["Eva", 3], **{prefix: "Guest "})
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται count_label(name, quantity = 1, prefix: ""). Γράψε call_from_data(arguments, options) που την καλεί ανοίγοντας positional Array και keyword Hash. Η υπογραφή της count_label καθορίζει τα αποδεκτά ορίσματα.

```ruby
call_from_data(["Eva"], {})
# => "Eva: 1"
```

```ruby
call_from_data(["Jo", 0], {prefix: "Guest "})
# => "Guest Jo: 0"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 119
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 119 --solution
```

Μια μικρή παραλλαγή

Κάλεσε τη μικρή μέθοδο format_pair(a, b:) με ένα positional και ένα keyword.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"X/Y"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

120 / ΜΑΘΗΜΑ

Ανάθεση σε πολλά ονόματα

Έννοια: Multiple assignment και rest destructuring

Πριν αρχίσεις: 011, 117

Η πολλαπλή ανάθεση μοιράζει στοιχεία σε ονόματα: `first, second = values`. Το `*middle` απορροφά τις ενδιάμεσες τιμές σε Array. Αν λείπουν τιμές, απλά ονόματα παίρνουν nil. Στην άσκηση ορίζουμε τουλάχιστον δύο στάσεις ώστε οι άκρες να είναι σαφείς.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
first, *middle, last = ["A", "B", "C", "D"]
[first, middle, last]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `route_parts(stops)`. Το stops είναι Array τουλάχιστον δύο Strings. Με destructuring δώσε {first: πρώτη, middle: ενδιάμεσο Array, last: τελευταία}. Μην αλλάξεις stops.

```ruby
route_parts(["A", "B", "C", "D"])
# => {first: "A", middle: ["B", "C"], last: "D"}
```

```ruby
route_parts(["A", "B"])
# => {first: "A", middle: [], last: "B"}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 120
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 120 --solution
```

Μια μικρή παραλλαγή

Αντάλλαξε δύο τοπικές τιμές με πολλαπλή ανάθεση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["B", "A"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

121 / ΜΑΘΗΜΑ

Δομή παραμέτρων block

Έννοια: Destructuring σε block parameters

Πριν αρχίσεις: 120, 014

Οι παράμετροι ενός block μπορούν να περιγράφουν τη δομή του στοιχείου. Το `|(name, seats), paid|` ανοίγει πρώτο nested ζευγάρι και δεύτερο πεδίο. Η ονομασία κοντά στην είσοδο μειώνει την ανάγκη για αλυσίδες δεικτών.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[["Eva", 2], ["Jo", 1]].map { |name, seats| "#{name}: #{seats}" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `nested_labels(records)`. Κάθε στοιχείο έχει μορφή [[name String, seats Integer], paid Boolean]. Με block destructuring δώσε "NAME: SEATS / PAID". Τα booleans εμφανίζονται ως true ή false. Διατήρησε σειρά.

```ruby
nested_labels([["Eva", 2], true], [["Jo", 0], false]])
# => ["Eva: 2 / true", "Jo: 0 / false"]
```

```ruby
nested_labels([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 121
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 121 --solution
```

Μια μικρή παραλλαγή

Από ζευγάρια ονόματος και ποσότητας κράτησε μόνο τα ονόματα με destructuring.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Eva", "Jo"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

122 / ΜΑΘΗΜΑ

Formatter με επιλογές · Εμπέδωση

Έννοια: Σύνθεση ορισμάτων, scope και προηγούμενων μετατροπών

Πριν αρχίσεις: 114, 118, 121

Ένας formatter γραμμής και ένας formatter αναφοράς μπορούν να μοιράζονται την ίδια σύμβαση keywords. Η αναφορά προωθεί επιλογές και δεν επαναλαμβάνει τη μορφοποίηση κάθε γραμμής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def decorate(text, prefix: "")
  "#{prefix}#{text}"
end
["A", "B"].map { |text| decorate(text, **{prefix: "* "}) }.join("\n")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε format_row(name, seats, prefix: "", uppercase: false) → "PREFIXNAME: SEATS", με προαιρετικά κεφαλαίο name. Γράψε format_table(rows, **options), όπου rows είναι ζευγάρια name,seats, που προωθεί options στη format_row και ενώνει με newline χωρίς τελικό newline.

```ruby
format_table([["Eva", 2], ["Jo", 1]], prefix: "* ", uppercase: true)
# => "* EVA: 2\n* JO: 1"
```

```ruby
format_row("Eva", 0)
# => "Eva: 0"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 122
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 122 --solution
```

Μια μικρή παραλλαγή

Μορφοποίησε ζευγάρια με δική σου lambda μέσω map.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["A=2", "B=3"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

123 / ΜΑΘΗΜΑ

Προεπιλεγμένη τιμή hash

Έννοια: `Hash.new(0)` και συμπεριφορά απόντος κλειδιού

Πριν αρχίσεις: 039, 059

Το `Hash.new(0)` επιστρέφει 0 όταν διαβάζεις απόν κλειδί, χωρίς να το εισάγει. Η `counts[key] += 1` ισοδυναμεί εδώ με ανάγνωση, πρόσθεση και ανάθεση. Έτσι η πρώτη ενημέρωση ξεκινά από μηδέν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
counts = Hash.new(0)
value = counts[:missing]
[value, counts.key?(:missing)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `manual_counts(categories)`. Για Array Strings χτίσε Hash.new(0) και αύξησε μετρητή για κάθε εμφάνιση. Επίστρεψε το Hash με default 0. Αν ο καλών διαβάσει άγνωστη κατηγορία, δεν πρέπει να προστεθεί κλειδί.

```ruby
manual_counts(["a", "b", "a"])
# => {"a" => 2, "b" => 1}
```

```ruby
h = manual_counts([])
value = h["x"]
[value, h.key?("x")]
# => [0, false]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 123
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 123 --solution
```

Μια μικρή παραλλαγή

Ξεκίνα από default 10 και πρόσθεσε 2 σε νέο κλειδί.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[12, 10, [:a]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

124 / ΜΑΘΗΜΑ

Προεπιλογή που κατασκευάζεται

Έννοια: Default block σε Hash

Πριν αρχίσεις: 035, 123

Το `Hash.new { |hash, key| ... }` εκτελεί block για απόν κλειδί. Για λίστες ανά όνομα γράφουμε `hash[key] = []`, ώστε κάθε κλειδί να αποκτήσει ξεχωριστό Array. Το `Hash.new([])` θα μοιραζόταν μία default λίστα και δεν θα αποθήκευε αυτόματα κλειδιά.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
notes = Hash.new { |hash, key| hash[key] = [] }
notes["Eva"] << "call"
notes["Jo"] << "email"
notes
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `notes_by_name(entries)`. Κάθε entry είναι [name String, note String]. Με default block χτίσε Hash λιστών σημειώσεων. Κράτησε σειρά ανά όνομα. Κάθε νέο κλειδί αποκτά ανεξάρτητο Array, και σε μεταγενέστερη ανάγνωση άγνωστου ονόματος.

RUBY
```ruby
notes_by_name([["A", "one"], ["B", "two"], ["A", "three"]])
# => {"A" => ["one", "three"], "B" => ["two"]}
```

RUBY
```ruby
h = notes_by_name([])
h["A"] << "x"
[h["A"], h["B"]]
# => [["x"], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 124
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 124 --solution
```

Μια μικρή παραλλαγή

Δείξε το πρόβλημα ενός κοινού default Array.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[["x"], []]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

125 / ΜΑΘΗΜΑ

Αποτέλεσμα που ενημερώνεται

Έννοια: `each_with_object`

Πριν αρχίσεις: 039, 121, 123

Η `each_with_object(initial)` δίνει στοιχείο και το ίδιο αντικείμενο συσσώρευσης στο block. Επιστρέφει το initial, ανεξάρτητα από την τελευταία τιμή του block. Ταιριάζει όταν μεταβάλλεις ένα Array ή Hash που κατασκευάζεις.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["A", "B"].each_with_object([]) { |name, result| result << "[#{name}]" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `products_index(products)`. Κάθε Hash έχει :code. Χτίσε νέο Hash code προς ολόκληρη εγγραφή. Σε διπλότυπο code επικρατεί η τελευταία εγγραφή. Χρησιμοποίησε each_with_object({}) και μην αλλάξεις τις αρχικές εγγραφές.

```ruby
products_index([{code: "A", price: 1}, {code: "B", price: 2}, {code: "A", price:
3}])
# => {"A" => {code: "A", price: 3}, "B" => {code: "B", price: 2}}
```

```ruby
products_index([])
# => {}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 125
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 125 --solution
```

Μια μικρή παραλλαγή

Χτίσε Array με τα θετικά στοιχεία χρησιμοποιώντας each_with_object.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 3]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

126 / ΜΑΘΗΜΑ

Αποτέλεσμα που περνά στο επόμενο βήμα

Έννοια: `reduce` / `inject` και αρχική τιμή

Πριν αρχίσεις: 008, 121, 125

Η `reduce(initial)` περνά στο block το τρέχον αποτέλεσμα και το επόμενο στοιχείο. Η τελική τιμή του block γίνεται το αποτέλεσμα του επόμενου βήματος. Η inject είναι εναλλακτικό όνομα. Στην κενή συλλογή επιστρέφεται η αρχική τιμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
[2, 3].reduce(10) { |balance, amount| balance - amount }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `apply_changes(initial, changes)`. Κάθε change είναι ["add", integer] ή ["multiply", integer]. Εφάρμοσε με reduce με τη δοσμένη σειρά τις πράξεις στο αρχικό integer. Επίστρεψε τελικό αποτέλεσμα. Όλες οι εντολές είναι έγκυρες.

RUBY
```ruby
apply_changes(2, [["add", 3], ["multiply", 4]])
# => 20
```

RUBY
```ruby
apply_changes(2, [["multiply", 4], ["add", 3]])
# => 11
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 126
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 126 --solution
```

Μια μικρή παραλλαγή

Με inject ένωσε τρία Strings με αρχικό κενό κείμενο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"ABC"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

127 / ΜΑΘΗΜΑ

Ανεξάρτητες αρχικές τιμές

Έννοια: `Array.new` με block και κοινές αναφορές

Πριν αρχίσεις: 035, 068, 124

Η `Array.new(count) { |index| ... }` καλεί το block ξεχωριστά για κάθε θέση. Ένα `[]` μέσα στο block δημιουργείται κάθε φορά. Αν γράψεις Array.new(count, []), όλες οι θέσεις αναφέρονται στο ίδιο Array.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
days = Array.new(2) { [] }
days[0] << "read"
days
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `day_lists(count)`. Για μη αρνητικό ακέραιο count επίστρεψε τόσα ανεξάρτητα κενά Arrays. Η προσθήκη εργασίας σε μία ημέρα δεν πρέπει να επηρεάζει άλλες.

```ruby
day_lists(3)
# => [[], [], []]
```

```ruby
days = day_lists(3)
days[0] << "read"
days
# => [["read"], [], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 127
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 127 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε τον index του block για ετικέτες Day 1 έως Day 3.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Day 1", "Day 2", "Day 3"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

128 / ΜΑΘΗΜΑ

Μνήμη προηγούμενου αποτελέσματος

Έννοια: Memoization και απουσία έναντι `false` / `nil`

Πριν αρχίσεις: 068, 108, 051

Memoization σημαίνει ότι θυμάσαι υπολογισμένο αποτέλεσμα. Αν false και nil είναι έγκυρες απαντήσεις, το ||= δεν αρκεί: θα ξαναϋπολόγιζε. Ένα ξεχωριστό @computed διακρίνει «υπολογίστηκε» από την ίδια την τιμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
cache = {answer: false}
if cache.key?(:answer)
  cache[:answer]
else
  "compute"
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε MemoCheck.new(&check). Η value καλεί το check μόνο στην πρώτη επιτυχημένη εκτέλεση και επιστρέφει το ίδιο αποθηκευμένο αποτέλεσμα σε επόμενες κλήσεις, ακόμη κι αν είναι nil ή false. Αν check προκαλέσει εξαίρεση, η επόμενη value ξαναδοκιμάζει.

RUBY
```ruby
calls = 0
memo = MemoCheck.new { calls += 1; false }
[memo.value, memo.value, calls]
# => [false, false, 1]
```

RUBY
```ruby
calls = 0
memo = MemoCheck.new { calls += 1; nil }
[memo.value, memo.value, calls]
# => [nil, nil, 1]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 128
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 128 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε Hash key? για cache όπου το αποθηκευμένο αποτέλεσμα είναι nil.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[nil, 1]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

129 / ΜΑΘΗΜΑ

Έλεγχος της επανάληψης

Έννοια: `next` και `break`, μαζί με τιμή αποτελέσματος

Πριν αρχίσεις: 013, 024

Το `next` τελειώνει μόνο την τρέχουσα εκτέλεση block και συνεχίζει με επόμενο στοιχείο. Το `break` σταματά την επανάληψη. Με `break value` ορίζεις αποτέλεσμα της κλήσης που διακόπτεται. Σε map, το next value δίνει τιμή για εκείνη τη θέση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[1, 4, 2].each do |n|
  break "found #{n}" if n >= 3
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `names_until_stop(lines)`. Για Array Strings κάνε strip. Παράλειψε κενά με next. Σταμάτησε στο ακριβές "STOP" με break, χωρίς να το βάλεις στην έξοδο. Κράτησε όσα προηγούμενα μη κενά ονόματα βρήκες. Μην αλλάξεις lines.

```ruby
names_until_stop([" A ", " ", "STOP", "B"])
# => ["A"]
```

```ruby
names_until_stop(["A", "B"])
# => ["A", "B"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 129
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 129 --solution
```

Μια μικρή παραλλαγή

Με map δώσε "skip" για το 0 με next και διπλάσιο για άλλους αριθμούς.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["skip", 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

130 / ΜΑΘΗΜΑ

Επανάληψη με συνθήκη

Έννοια: `while`, `until` και `loop`

Πριν αρχίσεις: 129, 011

Η while επαναλαμβάνει όσο η συνθήκη είναι truthy και η until όσο είναι falsy. Η loop επαναλαμβάνει block μέχρι break ή άλλο τερματισμό. Σε κάθε μορφή χρειάζεται σαφές βήμα προόδου. Χρησιμοποιούμε δείκτη για να αφήσουμε την αρχική λίστα ίδια.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
index = 0
values = []
while index < 3
  values << index
  index += 1
end
values
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `consume_tokens(tokens, limit)`. Tokens Array Strings και limit μη αρνητικός ακέραιος. Με while πάρε έως limit tokens ή μέχρι "END", όποιο συμβεί πρώτο. Το END δεν μπαίνει στην έξοδο. Μην καταναλώσεις το αρχικό Array.

RUBY
```ruby
consume_tokens(["A", "B", "C"], 2)
# => ["A", "B"]
```

RUBY
```ruby
consume_tokens(["A", "END", "B"], 9)
# => ["A"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 130
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 130 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε until για μέτρηση μέχρι 2 και loop με break για να επιστρέψεις "done".

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, "done"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

131 / ΜΑΘΗΜΑ

Συγκεκριμένα βήματα επανάληψης

Έννοια: `times`, `upto`, `downto`, `step`

Πριν αρχίσεις: 130, 012

Η `n.times` δίνει δείκτες 0 έως n-1. Η `upto` και η `downto` προχωρούν ανά ένα και περιλαμβάνουν το τελικό όριο. Η `step` επιτρέπει ρητό βήμα. Με block όλες εκτελούν την εργασία σου σε κάθε αριθμό.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = []
2.upto(4) { |n| values << n }
values
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `time_marks(start_minute, end_minute, step_size)`. Τα start_minute και end_minute είναι ακέραιοι με start <= end. Το step_size είναι θετικός ακέραιος. Με step δώσε Array από την αρχή έως όσα βήματα δεν ξεπερνούν το τέλος.

```ruby
time_marks(0, 10, 5)
# => [0, 5, 10]
```

```ruby
time_marks(2, 10, 3)
# => [2, 5, 8]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 131
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 131 --solution
```

Μια μικρή παραλλαγή

Φτιάξε αντίστροφη μέτρηση από 3 έως 1 και τρεις ετικέτες με times.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[[3, 2, 1], ["N1", "N2", "N3"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

132 / ΜΑΘΗΜΑ

Αρχή και υπόλοιπο

Έννοια: `take`, `drop`, `first(n)` και `last(n)`

Πριν αρχίσεις: 011

Η take(n) δίνει έως n αρχικά στοιχεία και η drop(n) όσα ακολουθούν. Η first(n) είναι επίσης αρχικό κομμάτι, ενώ last(n) τελικό. Με n=0 παίρνεις κενό αρχικό κομμάτι και ολόκληρο υπόλοιπο. Το first χωρίς n επιστρέφει στοιχείο, όχι Array.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
items = [1, 2, 3]
[items.take(2), items.drop(2)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `preview_queue(items, count)`. Με μη αρνητικό ακέραιο count επίστρεψε {preview: πρώτα count, remaining: υπόλοιπα}. Και τα δύο είναι Arrays. Μην αλλάξεις την είσοδο.

RUBY
```ruby
preview_queue([1, 2, 3], 2)
# => {preview: [1, 2], remaining: [3]}
```

RUBY
```ruby
preview_queue([1], 0)
# => {preview: [], remaining: [1]}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 132
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 132 --solution
```

Μια μικρή παραλλαγή

Δώσε πρώτο στοιχείο, Array πρώτου στοιχείου και δύο τελευταία.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[1, [1], [2, 3]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

133 / ΜΑΘΗΜΑ

Ομάδες σταθερού μεγέθους

Έννοια: `each_slice` με block

Πριν αρχίσεις: 013, 132

Η `each_slice(size)` με block δίνει διαδοχικά κομμάτια χωρίς επικάλυψη. Το τελευταίο μπορεί να είναι μικρότερο. Η παράμετρος του block είναι Array ολόκληρης της ομάδας. Το size πρέπει να είναι θετικό.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
groups = []
[1, 2, 3, 4, 5].each_slice(2) { |group| groups << group }
groups
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `teams_of_three(names)`. Με each_slice και block χώρισε το Array ονομάτων σε ομάδες των τριών. Κράτησε την τελευταία μικρή ομάδα. Επίστρεψε Array ομάδων χωρίς να αλλάξεις names.

RUBY
```ruby
teams_of_three(["A", "B", "C", "D"])
# => [["A", "B", "C"], ["D"]]
```

RUBY
```ruby
teams_of_three(["A", "B", "C"])
# => [["A", "B", "C"]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 133
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 133 --solution
```

Μια μικρή παραλλαγή

Χώρισε τέσσερα ποσά σε ζευγάρια και δώσε ένα άθροισμα ανά ζευγάρι.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[3, 7]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

134 / ΜΑΘΗΜΑ

Διαδοχικά ζευγάρια

Έννοια: `each_cons` με block

Πριν αρχίσεις: 133, 121

Η `each_cons(size)` δίνει επικαλυπτόμενα παράθυρα πλήρους μεγέθους. Για size 2 παίρνεις ζευγάρια γειτονικών τιμών. Αν υπάρχουν λιγότερα στοιχεία από size, το block δεν εκτελείται.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
pairs = []
[1, 4, 9].each_cons(2) { |pair| pairs << pair }
pairs
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `daily_changes(values)`. Για Array αριθμητικών μετρήσεων, δώσε τη διαφορά επόμενης μείον προηγούμενη για κάθε γειτονικό ζευγάρι. Χρησιμοποίησε each_cons(2) με block. Για 0 ή 1 μέτρηση δώσε [].

```ruby
daily_changes([3, 8, 6])
# => [5, -2]
```

```ruby
daily_changes([4, 4])
# => [0]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 134
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 134 --solution
```

Μια μικρή παραλλαγή

Αντί για διαφορές, δώσε ετικέτες μεταβάσεων στάσεων.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["A → B", "B → C"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

135 / ΜΑΘΗΜΑ

Συλλογές δίπλα δίπλα

Έννοια: `zip`

Πριν αρχίσεις: 011, 121

Η `names.zip(scores)` συνδέει στοιχεία ανά θέση σε Arrays. Το μήκος της εξόδου είναι του receiver names. Αν scores είναι κοντύτερο συμπληρώνεται nil, αν είναι μακρύτερο οι επιπλέον τιμές του αγνοούνται.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["A", "B", "C"].zip([8, 9])
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `name_scores(names, scores)`. Δώσε Array Hashes {name: όνομα, score: αντίστοιχος βαθμός ή nil}. Η σειρά και το πλήθος καθορίζονται από names. Μην αλλάξεις καμία λίστα.

```ruby
name_scores(["A", "B"], [0])
# => [{name: "A", score: 0}, {name: "B", score: nil}]
```

```ruby
name_scores(["A"], [1, 2])
# => [{name: "A", score: 1}]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 135
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 135 --solution
```

Μια μικρή παραλλαγή

Σύνδεσε τρεις παράλληλες λίστες με μία zip.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[["A", 1, true], ["B", 2, false]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

136 / ΜΑΘΗΜΑ

Σελίδες και γειτονικές τιμές · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 132, 133, 134

Διάκρινε προεπισκόπηση, μη επικαλυπτόμενες ομάδες και γειτονικές μεταβολές. Κάθε αποτέλεσμα περιγράφει διαφορετική σχέση ανάμεσα στα ίδια δεδομένα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = [2, 5, 9]
[values.take(2), values.drop(2)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `sequence_summary(values)`. Για Array ακεραίων δώσε {preview: δύο πρώτα, groups: ομάδες των δύο, changes: γειτονικές διαφορές}. Κράτησε το τελευταίο μικρό group και άφησε την είσοδο ίδια.

```ruby
sequence_summary([2, 5, 9])
# => {preview: [2, 5], groups: [[2, 5], [9]], changes: [3, 4]}
```

```ruby
sequence_summary([2])
# => {preview: [2], groups: [[2]], changes: []}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 136
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 136 --solution
```

Μια μικρή παραλλαγή

Πάρε τις πρώτες τρεις τιμές και δώσε γειτονικά ζευγάρια μόνο αυτών.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[[1, 2], [2, 3]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

137 / ΜΑΘΗΜΑ

Πρόγραμμα ομάδων · Εμπέδωση

Έννοια: Σύνθεση `zip`, `each_slice`, `each_cons` και προηγούμενης ύλης

Πριν αρχίσεις: 133, 134, 135

Ένωσε παράλληλες λίστες πριν φτιάξεις ομάδες. Οι μεταβάσεις χρειάζονται διαδοχικά ζευγάρια από τις ονομασμένες συναντήσεις. Οι κανόνες δίνονται έτοιμοι.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
pairs = ["Eva", "Jo"].zip([1, 2])
pairs.map { |name, seat| "#{name}@#{seat}" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `group_schedule(names, seats, meetings)`. Names και seats έχουν ίδιο μήκος. Φτιάξε ομάδες των δύο από ζευγάρια [name, seat]. Από meetings Array Strings δώσε μεταβάσεις "A → B". Επίστρεψε {groups: ομάδες, transitions: μεταβάσεις}, χωρίς μεταβολή εισόδων.

```ruby
group_schedule(["A", "B", "C"], [1, 2, 3], ["Intro", "Practice", "Review"])
# => {groups: [["A", 1], ["B", 2]], [["C", 3]]], transitions: ["Intro → Practice",
"Practice → Review"]}
```

```ruby
group_schedule([], [], [])
# => {groups: [], transitions: []}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 137
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 137 --solution
```

Μια μικρή παραλλαγή

Δώσε πλήθος ατόμων σε ήδη έτοιμες ομάδες.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 1]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

138 / ΜΑΘΗΜΑ

Η επανάληψη ως αντικείμενο

Έννοια: `each` και `File.foreach` χωρίς block, `Enumerator`

Πριν αρχίσεις: 013, 096, 108

Πολλές μέθοδοι επανάληψης χωρίς block επιστρέφουν Enumerator. Είναι αντικείμενο που περιγράφει επανάληψη για αργότερα. Μπορείς να του δώσεις block με each ή map. Η δημιουργία File.foreach χωρίς block αναβάλλει την ανάγνωση μέχρι τη διάσχιση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
enumerator = ["A", "B"].each
[enumerator.class, enumerator.map { |name| "[#{name}]" }]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `name_enumerator(names)`. Επίστρεψε Enumerator από names.each χωρίς block. Δεν χρειάζεται δικός σου generator. Όταν ο καλών κάνει map, πρέπει να πάρει τα αρχικά ονόματα με σειρά.

```ruby
e = name_enumerator(["Eva", "Jo"])
[e.class, e.map(&:upcase)]
# => [Enumerator, ["EVA", "JO"]]
```

```ruby
name_enumerator([]).to_a
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 138
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 138 --solution
```

Μια μικρή παραλλαγή

Πάρε Enumerator από δοσμένο αρχείο χωρίς block και μετά καθάρισε τις γραμμές με map.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Eva", "Jo"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

139 / ΜΑΘΗΜΑ

Ζήτησε το επόμενο στοιχείο

Έννοια: `next` και `StopIteration`

Πριν αρχίσεις: 082, 138

Η Enumerator#next επιστρέφει το επόμενο στοιχείο και προχωρά την εξωτερική θέση. Όταν εξαντληθεί η πηγή προκαλεί StopIteration. Ένα πραγματικό nil στοιχείο παραμένει κανονικό αποτέλεσμα, άρα δεν μπορεί να χρησιμεύσει ως μοναδικό σήμα τέλους.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
e = ["A", "B"].each
[e.next, e.next]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `next_record(enumerator)`. Ζήτησε ένα στοιχείο με next. Δώσε {done: false, value: στοιχείο}, ακόμη και για nil. Σε StopIteration δώσε {done: true, value: nil}. Ο Enumerator επιτρέπεται να προχωρήσει.

```ruby
next_record(["Eva"].each)
# => {done: false, value: "Eva"}
```

```ruby
e = [nil].each
[next_record(e), next_record(e)]
# => [{done: false, value: nil}, {done: true, value: nil}]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 139
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 139 --solution
```

Μια μικρή παραλλαγή

Κατανάλωσε μία τιμή και πιάσε την εξάντληση στην επόμενη κλήση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"finished"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

140 / ΜΑΘΗΜΑ

Κοίτα χωρίς να προχωρήσεις

Έννοια: `peek` και `rewind`

Πριν αρχίσεις: 139

Η `peek` δείχνει την επόμενη τιμή χωρίς να μετακινεί την εξωτερική θέση. Επαναλαμβανόμενα peek δίνουν την ίδια τιμή. Η `rewind` επαναφέρει την επανάληψη στην αρχή όταν το υποστηρίζει η πηγή, όπως ένα Array. Η peek σε εξάντληση προκαλεί StopIteration.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
e = ["A", "B"].each
[e.peek, e.peek, e.next, e.next]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `peek_restart(values)`. Η είσοδος είναι μη κενό Array. Φτιάξε δικό του Enumerator, πάρε peek, μετά next, μετά rewind και ξανά next. Επίστρεψε τις τρεις τιμές σε Array. Μην αλλάξεις values.

```ruby
peek_restart(["A", "B"])
# => ["A", "A", "A"]
```

```ruby
peek_restart([nil, "A"])
# => [nil, nil, nil]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 140
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 140 --solution
```

Μια μικρή παραλλαγή

Κατανάλωσε και τις δύο τιμές ενός Enumerator, κάνε rewind και ζήτησε ξανά την πρώτη.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `1`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

141 / ΜΑΘΗΜΑ

Προσθήκη θέσης σε μια ακολουθία

Έννοια: `with_index` και προσαρμοσμένη αρχή

Πριν αρχίσεις: 022, 138

Η `map` χωρίς block δίνει Enumerator. Πάνω του η `with_index(start)` προσθέτει δεύτερη τιμή στο block. Έτσι `map.with_index(1)` κρατά τη μετατροπή της map και δίνει ανθρώπινη αρίθμηση από 1. Η each_with_index αρχίζει πάντα από 0.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
["A", "B"].map.with_index(5) { |name, number| "#{number}: #{name}" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `indexed_labels(names, start = 1)`. Με map.with_index(start) δώσε Strings "N. NAME". Το start είναι ακέραιος, ακόμη και 0 ή αρνητικός. Κράτησε σειρά και μη μεταβάλεις names.

RUBY
```ruby
indexed_labels(["Eva", "Jo"])
# => ["1. Eva", "2. Jo"]
```

RUBY
```ruby
indexed_labels(["A", "B"], -1)
# => ["-1. A", "0. B"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 141
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 141 --solution
```

Μια μικρή παραλλαγή

Δώσε ζευγάρια [τιμή, θέση] από Enumerator με αρχή 10.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[["A", 10], ["B", 11]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

142 / ΜΑΘΗΜΑ

Πρόβλεψη της επόμενης τιμής · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 139, 140, 141

Παρακολούθησε τη θέση του Enumerator σε κάθε βήμα: peek δεν καταναλώνει, next καταναλώνει, rewind ξαναρχίζει. Η άσκηση έχει τουλάχιστον δύο τιμές ώστε όλη η προβλεπόμενη ακολουθία να είναι ορισμένη.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
e = [2, 4].each
[e.next, e.peek, e.peek]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `enumerator_trace(values)`. Για Array τουλάχιστον δύο στοιχείων φτιάξε Enumerator. Κατέγραψε με σειρά peek, next, peek, next, μετά rewind και next. Επίστρεψε Array των πέντε τιμών. Η είσοδος μένει ίδια.

```ruby
enumerator_trace(["A", "B", "C"])
# => ["A", "A", "B", "B", "A"]
```

```ruby
enumerator_trace([false, nil])
# => [false, false, nil, nil, false]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 142
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 142 --solution
```

Μια μικρή παραλλαγή

Με next και rescue κατανάλωσε πεπερασμένο Enumerator με πραγματικό nil στη μέση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[1, nil, 2]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

143 / ΜΑΘΗΜΑ

Έλεγχος μορφής κειμένου

Έννοια: Regexp literal, `match?`, anchors και character classes

Πριν αρχίσεις: 030

Το `/.../` δημιουργεί Regexp. Η match? δίνει boolean. Τα `[A-Z]` και `[0-9]` περιγράφουν έναν χαρακτήρα από συγκεκριμένο σύνολο. Τα `\A` και `\z` ορίζουν αρχή και τέλος όλου του String, όχι απλώς γραμμής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
/\A[A-Z][0-9]\z/.match?("B4")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `valid_code?(text)`. Δέχεται String. Δώσε true μόνο για ακριβώς δύο κεφαλαία ASCII γράμματα και τρία ASCII ψηφία, π.χ. AB123. Χωρίς κενά ή τελικό newline. Σε αυτό το βήμα γράψε τις θέσεις ρητά, χωρίς quantifiers.

```ruby
valid_code?("AB123")
# => true
```

```ruby
valid_code?("Ab123")
# => false
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 143
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 143 --solution
```

Μια μικρή παραλλαγή

Δέξου κωδικό ενός πεζού γράμματος και ενός ψηφίου.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

144 / ΜΑΘΗΜΑ

Ομάδες και επαναλήψεις σε regex

Έννοια: Quantifiers, alternation και escaping

Πριν αρχίσεις: 143

Τα `{n}` και `{min,max}` δηλώνουν πλήθος επαναλήψεων, το + μία ή περισσότερες και το ? προαιρετική εμφάνιση. Το `(?:A|B)` ομαδοποιεί εναλλακτικές χωρίς capture. Το backslash κάνει ειδικό χαρακτήρα κυριολεκτικό, π.χ. `\.`. Για δοσμένο κείμενο υπάρχει Regexp.escape.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
/\A(?:X|Y)-[0-9]{2}\z/.match?("Y-04")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `valid_batch_code?(text)`. Δέξου μόνο AB ή CD, παύλα και 1 έως 3 ψηφία. Ολόκληρο το String πρέπει να ταιριάζει. Τα αρχικά μηδενικά επιτρέπονται.

```ruby
valid_batch_code?("CD-007")
# => true
```

```ruby
valid_batch_code?("AB-0")
# => true
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 144
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 144 --solution
```

Μια μικρή παραλλαγή

Δέξου το κυριολεκτικό όνομα file.txt, όχι fileXtxt.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[true, false]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

145 / ΜΑΘΗΜΑ

Εξαγωγή ονομασμένων πεδίων

Έννοια: `MatchData`, captures και named captures

Πριν αρχίσεις: 037, 143, 144

Η Regexp#match επιστρέφει MatchData ή nil. Το capture `(?<name>...)` δίνει όνομα σε τμήμα και διαβάζεται με match[:name]. Το match[0] είναι ολόκληρη η αντιστοίχιση. Όλα τα πεδία captures αρχίζουν ως Strings.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
match = /\A(?<code>[A-Z]{2}):(?<count>[0-9]+)\z/.match("XY:12")
[match[0], match[:code], match[:count]]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `parse_code_quantity(text)`. Έγκυρη μορφή: δύο κεφαλαία ASCII γράμματα, : και ένα ή περισσότερα ψηφία. Δώσε {code: String, quantity: Integer} με named captures ή nil για άκυρο String.

RUBY
```ruby
parse_code_quantity("AB:007")
# => {code: "AB", quantity: 7}
```

RUBY
```ruby
parse_code_quantity("CD:0")
# => {code: "CD", quantity: 0}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 145
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 145 --solution
```

Μια μικρή παραλλαγή

Με αριθμημένα captures χώρισε δύο λέξεις γύρω από /.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["A", "BC"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

146 / ΜΑΘΗΜΑ

Πολλά ταιριάσματα

Έννοια: `scan`

Πριν αρχίσεις: 144, 145

Η `scan` βρίσκει μη επικαλυπτόμενες αντιστοιχίσεις σε όλο το String. Χωρίς captures επιστρέφει Strings αντιστοιχίσεων. Με captures επιστρέφει Array πεδίων ανά αντιστοίχιση, ακόμη κι αν υπάρχει μόνο ένα capture.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
"A12 B3".scan(/[A-Z][0-9]+/)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `template_markers(text)`. Βρες όλα τα tokens {name} όπου name περιέχει ένα ή περισσότερα πεζά ASCII γράμματα ή underscore. Επίστρεψε μόνο τα ονόματα, με σειρά και επαναλήψεις. Άλλοι χαρακτήρες μέσα σε braces δεν ταιριάζουν.

```ruby
template_markers("Hi {name}, {date}, {name}")
# => ["name", "date", "name"]
```

```ruby
template_markers("{A} {} {x1} {valid_key}")
# => ["valid_key"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 146
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 146 --solution
```

Μια μικρή παραλλαγή

Χωρίς captures, βρες όλους τους ακέραιους ως Strings από ένα κείμενο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["12", "3"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

147 / ΜΑΘΗΜΑ

Αντικατάσταση με υπολογισμό

Έννοια: `gsub` με Regexp και block

Πριν αρχίσεις: 038, 144, 146

Η gsub με Regexp και block καλεί το block για κάθε ταιριασμένο String. Η επιστροφή του block γίνεται το replacement. Μπορείς να χρησιμοποιήσεις Hash δεδομένων αντί να κατασκευάζεις χειροκίνητα όλα τα μηνύματα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
"A2 B3".gsub(/[0-9]+/) { |digits| (Integer(digits, 10) * 2).to_s }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `render_markers(text, values)`. Για tokens {key} με πεζά ASCII γράμματα/underscore αντικατάστησε με values[key], όπου τα Hash κλειδιά και οι τιμές είναι Strings. Αν λείπει το κλειδί, κράτησε το αρχικό token. Μην αλλάξεις text ή values.

```ruby
render_markers("Hi {name}: {count}", {"name" => "Eva"})
# => "Hi Eva: {count}"
```

```ruby
render_markers("A{gap}B", {"gap" => ""})
# => "AB"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 147
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 147 --solution
```

Μια μικρή παραλλαγή

Με block βάλε αγκύλες γύρω από κάθε αριθμό.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"A[12] B[3]"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

148 / ΜΑΘΗΜΑ

Κείμενο σε δομημένες τιμές · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 062, 145, 147

Χρησιμοποίησε αυστηρό pattern για μία εγγραφή και άφησε τις έτοιμες μεθόδους να συλλέξουν αποδεκτές γραμμές. Κάθε επιτυχία έχει δύο καθαρά πεδία που μπαίνουν σε τελική ετικέτα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
/\A(?<name>[A-Za-z]+):(?<count>[0-9]+)\z/.match("Eva:2")[:name]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `parsed_labels(lines)`. Κράτησε μόνο γραμμές NAME:COUNT με NAME ένα ή περισσότερα ASCII γράμματα και COUNT ψηφία. Δώσε "NAME (COUNT)" με αριθμητικά κανονικοποιημένο COUNT. Άκυρες γραμμές απορρίπτονται. Σειρά διατηρείται.

```ruby
parsed_labels(["Eva:02", "bad", "Jo:0", "A:-1"])
# => ["Eva (2)", "Jo (0)"]
```

```ruby
parsed_labels([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 148
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 148 --solution
```

Μια μικρή παραλλαγή

Αντικατάστησε το σταθερό token {total} με δοσμένο πλήθος μέσω gsub block.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Count: 3"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

149 / ΜΑΘΗΜΑ

Χρονικές στιγμές και διάρκειες

Έννοια: `Time`, offsets και αφαίρεση χρονικών στιγμών

Πριν αρχίσεις: 090, 002

Το Time παριστάνει χρονική στιγμή. Με `require "time"` και Time.iso8601 διαβάζεις ISO κείμενο με ρητό offset. Η αφαίρεση δύο στιγμών δίνει διάρκεια σε δευτερόλεπτα. Ίδιες τοπικές ώρες με διαφορετικά offsets δεν είναι ίδια στιγμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
require "time"
a = Time.iso8601("2026-01-10T10:00:00+02:00")
b = Time.iso8601("2026-01-10T08:00:00Z")
a - b
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `meeting_minutes(start_text, finish_text)`. Τα έγκυρα ISO8601 Strings έχουν ρητό offset, χωρίς κλάσματα δευτερολέπτου. Finish >= start και διαφορά ακέραιων λεπτών. Επίστρεψε διάρκεια σε ακέραια λεπτά, ανεξάρτητα από τα offsets.

RUBY
```ruby
meeting_minutes("2026-01-10T10:00:00+02:00", "2026-01-10T09:30:00Z")
# => 90
```

RUBY
```ruby
meeting_minutes("2026-01-10T10:00:00+02:00", "2026-01-10T08:00:00Z")
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 149
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 149 --solution
```

Μια μικρή παραλλαγή

Μετέτρεψε στιγμή σε UTC με getutc και δώσε ISO String.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"2026-01-10T08:00:00Z"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

150 / ΜΑΘΗΜΑ

Ημερολογιακές ημερομηνίες

Έννοια: `Date`, parsing και μορφοποίηση

Πριν αρχίσεις: 090, 149

Η Date περιγράφει ημερολογιακή ημέρα χωρίς ώρα. Με Date.iso8601 διαβάζεις YYYY-MM-DD. Η strftime μορφοποιεί: %d ημέρα, %m μήνας, %Y έτος. Η πρόσθεση ακεραίου σε Date μετακινεί τόσες ημερολογιακές ημέρες.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "date"
(Date.iso8601("2024-02-28") + 1).strftime("%d/%m/%Y")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `event_dates(dates)`. Δέχεται Array έγκυρων ημερομηνιών YYYY-MM-DD. Δώσε Array ετικετών DD/MM/YYYY με αρχική σειρά. Δεν χρειάζεται έλεγχος άκυρων ημερομηνιών.

```ruby
event_dates(["2024-02-29", "2026-01-05"])
# => ["29/02/2024", "05/01/2026"]
```

```ruby
event_dates([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 150
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 150 --solution
```

Μια μικρή παραλλαγή

Πρόσθεσε μία ημέρα στην τελευταία ημέρα του έτους.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"2026-01-01"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

151 / ΜΑΘΗΜΑ

Σύνολα τιμών

Έννοια: `Set`, ένωση, τομή και διαφορά

Πριν αρχίσεις: 056, 090

Το Set κρατά μοναδικές τιμές. Με `|` παίρνεις ένωση, με `&` τομή και με `-` διαφορά. Η include? ελέγχει συμμετοχή. Χρησιμοποίησε to_a.sort όταν θέλεις σταθερή ταξινομημένη παρουσίαση αντί για την έννοια συνόλου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
require "set"
(Set.new(["A", "B"]) & Set.new(["B", "C"])).to_a.sort
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `compare_groups(left, right)`. Δύο Arrays ASCII ονομάτων. Επίστρεψε {all: ένωση, common: τομή, only_left: αριστερή διαφορά}, με ταξινομημένα μοναδικά Arrays ως τιμές. Μην αλλάξεις εισόδους.

RUBY
```ruby
compare_groups(["B", "A", "A"], ["B", "C"])
# => {all: ["A", "B", "C"], common: ["B"], only_left: ["A"]}
```

RUBY
```ruby
compare_groups([], [])
# => {all: [], common: [], only_left: []}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 151
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 151 --solution
```

Μια μικρή παραλλαγή

Έλεγξε αν το όνομα Eva ανήκει σε Set και αν τα διπλότυπα αυξάνουν το μέγεθος.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[true, 1]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

152 / ΜΑΘΗΜΑ

Ακριβείς δεκαδικές τιμές

Έννοια: `BigDecimal` και κατασκευή από String

Πριν αρχίσεις: 021, 090

Η BigDecimal κατασκευάζεται από δεκαδικό String ώστε να μην εισαχθεί πρώτα η προσέγγιση Float. Η round(2, BigDecimal::ROUND_HALF_UP) δίνει ρητό κανόνα για δύο δεκαδικές θέσεις. Η to_s("F") δίνει δεκαδικό κείμενο χωρίς επιστημονική γραφή, αλλά δεν συμπληρώνει πάντα δύο μηδενικά.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "bigdecimal"
(BigDecimal("0.1") + BigDecimal("0.2")).to_s("F")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `decimal_charge(amounts, rate)`. Amounts Array έγκυρων δεκαδικών Strings και rate έγκυρο δεκαδικό String. Άθροισε amount * rate σε BigDecimal, στρογγύλεψε μόνο το τελικό σύνολο σε δύο θέσεις με HALF_UP και δώσε to_s("F"). Κενό amounts δίνει "0.0".

```ruby
decimal_charge(["0.1", "0.2"], "1")
# => "0.3"
```

```ruby
decimal_charge(["1.005"], "1")
# => "1.01"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 152
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 152 --solution
```

Μια μικρή παραλλαγή

Στρογγύλεψε αρνητικό μισό στην ίδια πολιτική HALF_UP.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"-1.01"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

153 / ΜΑΘΗΜΑ

Επιλογές εντολής

Έννοια: `OptionParser`

Πριν αρχίσεις: 048, 099, 108

Η OptionParser δηλώνει flags με on και εκτελεί αντίστοιχο block. Η parse! καταναλώνει από το Array τα αναγνωρισμένα options και αφήνει τα υπόλοιπα ορίσματα. Δούλεψε σε dup όταν ο καλών χρειάζεται το αρχικό args.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "optparse"
options = {upper: false}
args = ["--upper", "Eva"]
OptionParser.new { |p| p.on("--upper") { options[:upper] = true } }.parse!(args)
[options, args]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `parse_format_options(args)`. Αναγνώρισε --upper χωρίς τιμή και --prefix TEXT με String τιμή. Defaults {upper: false, prefix: ""}. Επίστρεψε {options: Hash, arguments: υπόλοιπο Array}. Μην αλλάξεις args. Άγνωστες επιλογές αφήνουν το error της OptionParser να διαδοθεί.

```ruby
parse_format_options(["--upper", "--prefix", "Hi ", "Eva"])
# => {options: {upper: true, prefix: "Hi "}, arguments: ["Eva"]}
```

```ruby
parse_format_options(["Eva"])
# => {options: {upper: false, prefix: ""}, arguments: ["Eva"]}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 153
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 153 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε -- για να περάσεις όνομα που αρχίζει με -- ως απλό όρισμα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["--name"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

154 / ΜΑΘΗΜΑ

Εκτέλεση άλλου τοπικού προγράμματος

Έννοια: `Process` / `Open3`, exit status και ορίσματα ως ξεχωριστές τιμές

Πριν αρχίσεις: 099, 100, 097

Η Open3.capture3 επιστρέφει stdout, stderr και Process::Status. Δώσε executable και ορίσματα ως χωριστές τιμές, ώστε το κείμενο να περάσει ως όρισμα χωρίς shell ερμηνεία. Η RbConfig.ruby δίνει τον τρέχοντα Ruby interpreter. Με Process.spawn παίρνεις pid και με Process.wait2 περιμένεις και συλλέγεις status.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "open3"
require "rbconfig"
out, err, status = Open3.capture3(RbConfig.ruby, __dir__ + "/support/worker.rb",
"Eva")
[out, err, status.exitstatus]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `run_local_worker(name)`. Τρέξε μόνο το δοσμένο support/worker.rb με τον τρέχοντα Ruby interpreter και ένα String όρισμα name. Δώσε {output: stdout, error: stderr, status: ακέραιος exitstatus}. Το fixture για "fail" γράφει "failed" στο stderr και τερματίζει με 2, αλλιώς χαιρετά. Δεν εκτελείται κείμενο από το name.

```ruby
run_local_worker("Eva Jo")
# => {output: "Hello Eva Jo\n", error: "", status: 0}
```

```ruby
run_local_worker("fail")
# => {output: "", error: "failed\n", status: 2}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 154
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 154 --solution
```

Μια μικρή παραλλαγή

Με Process.spawn και wait2 εκτέλεσε το ίδιο fixture για Jo, κατευθύνοντας stdout σε tmp/child.txt.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[true, true, "Hello Jo\n"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

155 / ΜΑΘΗΜΑ

Μετατροπή τοπικών δεδομένων · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 094, 101, 150, 152

Σύνδεσε δύο τύπους δεδομένων που έχουν ήδη δικές τους έτοιμες λειτουργίες: ημερομηνίες και δεκαδικά ποσά. Δεν χρειάζεται χειροκίνητη διάσπαση ημερομηνιών ή αριθμητική με Float.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
require "date"
require "bigdecimal"
"#{Date.iso8601("2026-03-02").strftime("%d/%m/%Y")}:
#{BigDecimal("2.50").to_s("F")}"
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `dated_amount_report(path)`. Το δοσμένο UTF-8 JSON αρχείο έχει Array εγγραφών με "date" έγκυρο YYYY-MM-DD και "amount" έγκυρο δεκαδικό String. Επίστρεψε Array `DD/MM/YYYY: AMOUNT`, με amount στρογγυλοποιημένο HALF_UP σε δύο θέσεις και to_s("F"). Κράτησε σειρά.

```ruby
dated_amount_report(__dir__ + "/fixtures/rows.json")
# => ["29/02/2024: 1.01", "01/01/2026: 0.0"]
```

```ruby
dated_amount_report(__dir__ + "/fixtures/empty.json")
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 155
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 155 --solution
```

Μια μικρή παραλλαγή

Μετάφερε μία δοσμένη Date κατά δύο ημέρες πριν τη μορφοποίηση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"02/02/2026"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

156 / ΜΑΘΗΜΑ

Προστασία από μεταβολή

Έννοια: `freeze`, `frozen?` και ρηχή ακινητοποίηση

Πριν αρχίσεις: 035, 037

Η freeze απαγορεύει μεταβολές στο συγκεκριμένο αντικείμενο και το επιστρέφει. Η frozen? ελέγχει την κατάσταση. Το freeze είναι ρηχό: ένα παγωμένο Array ή Hash μπορεί να περιέχει μεταβλητά Strings ή Arrays.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
values = ["A"].freeze
values[0] << "!"
[values, values.frozen?, values[0].frozen?]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `frozen_settings(settings)`. Δώσε παγωμένο ρηχό αντίγραφο του settings Hash. Η αρχική εξωτερική δομή πρέπει να παραμένει μεταβλητή. Τα εσωτερικά αντικείμενα επιτρέπεται να είναι κοινά. Μην κάνεις deep freeze.

RUBY
```ruby
frozen_settings({active: true})
# => {active: true}
```

RUBY
```ruby
h = {labels: []}
copy = frozen_settings(h)
[copy.frozen?, h.frozen?, copy[:labels].equal?(h[:labels])]
# => [true, false, true]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 156
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 156 --solution
```

Μια μικρή παραλλαγή

Πάγωσε ρητά και το εσωτερικό String πριν παγώσεις το Array.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[true, true]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

157 / ΜΑΘΗΜΑ

Ισότητα και ταυτότητα

Έννοια: `==`, `eql?`, `equal?` και συμβατό `hash`

Πριν αρχίσεις: 068, 056, 156

Η == δηλώνει ισότητα τιμής σύμφωνα με τον τύπο. Η eql? μαζί με hash καθορίζει ισότητα κλειδιών Hash και στοιχείων uniq. Ίσες κατά eql? τιμές πρέπει να έχουν ίδιο hash. Η equal? ελέγχει αν είναι το ίδιο αντικείμενο. Π.χ. 1 == 1.0 αλλά όχι 1.eql?(1.0).

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
a = "A"
b = "A"
[a == b, a.eql?(b), a.equal?(b), 1 == 1.0, 1.eql?(1.0)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε CodeValue.new(code String), με reader code. Αποθήκευσε παγωμένο αντίγραφο. Δύο CodeValue είναι ίσα με == και eql? όταν έχουν ίδιο code. Άλλοι τύποι δεν είναι ίσοι. Δώσε συμβατό hash ώστε να λειτουργούν ως Hash keys και σε uniq.

```ruby
a = CodeValue.new("A")
b = CodeValue.new("A")
[a == b, a.eql?(b), a.equal?(b)]
# => [true, true, false]
```

```ruby
a = CodeValue.new("A")
b = CodeValue.new("A")
[{a => 7}[b], [a, b].uniq.length]
# => [7, 1]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 157
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 157 --solution
```

Μια μικρή παραλλαγή

Σύγκρινε 1 και 1.0 ως κλειδιά Hash.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `2`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

158 / ΜΑΘΗΜΑ

Μικρή εγγραφή χωρίς πολύ boilerplate

Έννοια: `Struct`

Πριν αρχίσεις: 068, 069, 048

Η Struct.new δημιουργεί μικρή κλάση εγγραφής με readers, writers και αρχικοποίηση. Με keyword_init: true καλείς new με ονομασμένα πεδία. Το block της Struct.new προσθέτει μεθόδους στην παραγόμενη κλάση. Τα Structs παραμένουν μεταβλητά.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
DemoRow = Struct.new(:name, :seats, keyword_init: true)
row = DemoRow.new(name: "Eva", seats: 2)
[row.name, row.seats]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Όρισε ReportRow με Struct για πεδία name, quantity, price και keyword_init: true. Πρόσθεσε total που επιστρέφει quantity * price. Οι αριθμοί μη αρνητικοί ακέραιοι. Η quantity πρέπει να παραμένει μεταβλητή μέσω setter.

RUBY
```ruby
r = ReportRow.new(name: "Tea", quantity: 3, price: 2)
[r.name, r.total]
# => ["Tea", 6]
```

RUBY
```ruby
r = ReportRow.new(name: "Tea", quantity: 3, price: 2)
r.quantity = 0
r.total
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 158
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 158 --solution
```

Μια μικρή παραλλαγή

Όρισε Struct δύο positional πεδίων και φτιάξε μία παρουσία.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 3]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

159 / ΜΑΘΗΜΑ

Τιμή με σταθερά μέλη

Έννοια: `Data.define` και `with`

Πριν αρχίσεις: 156, 158

Η `Data.define(:field, ...)` δημιουργεί τύπο τιμής χωρίς writers. Η `with(field: new_value)` φτιάχνει παραλλαγή με αλλαγμένα πεδία. Η ακινητοποίηση είναι ρηχή: μεταβλητά αντικείμενα μέσα στα πεδία μπορούν ακόμη να αλλάξουν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
ViewSettings = Data.define(:theme, :size)
original = ViewSettings.new(theme: "light", size: 2)
changed = original.with(size: 3)
[original.size, changed.size]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Όρισε DisplaySettings ως Data με theme και tags. Γράψε change_theme(settings, theme) με with. Το αρχικό theme μένει ίδιο και τα tags επιτρέπεται να είναι κοινό Array μεταξύ αρχικού και νέου αντικειμένου.

```ruby
a = DisplaySettings.new(theme: "light", tags: [])
b = change_theme(a, "dark")
[a.theme, b.theme, a.equal?(b)]
# => ["light", "dark", false]
```

```ruby
a = DisplaySettings.new(theme: "light", tags: [])
b = change_theme(a, "dark")
b.tags << "new"
[a.tags, b.tags]
# => [["new"], ["new"]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 159
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 159 --solution
```

Μια μικρή παραλλαγή

Κράτησε εσωτερικό Array παγωμένο πριν το βάλεις σε Data.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[true, true]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

160 / ΜΑΘΗΜΑ

Συμπεριφορά πάνω σε ένα αντικείμενο

Έννοια: `extend`

Πριν αρχίσεις: 079, 073

Η extend προσθέτει τις instance methods ενός module ως μεθόδους σε ένα συγκεκριμένο αντικείμενο. Αν το αντικείμενο είναι κλάση, οι μέθοδοι καλούνται πάνω στην κλάση. Η include αντίστοιχα δίνει συμπεριφορά στις παρουσίες της κλάσης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
module DemoFormat
  def bracket(text)
    "[#{text}]"
  end
end
object = Object.new
object.extend(DemoFormat)
object.bracket("A")
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται LabelTools με label(name) → "[NAME]". Γράψε enable_labels(object) που κάνει extend του module μόνο στο δοσμένο αντικείμενο και επιστρέφει το ίδιο αντικείμενο. Άλλο Object.new πρέπει να παραμένει χωρίς label.

```ruby
enable_labels(Object.new).label("Eva")
# => "[Eva]"
```

```ruby
o = Object.new
enable_labels(o).equal?(o)
# => true
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 160
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 160 --solution
```

Μια μικρή παραλλαγή

Κάνε extend σε κλάση ώστε να αποκτήσει μέθοδο κλάσης.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Ready"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

161 / ΜΑΘΗΜΑ

Σύντομες μορφές εκφράσεων

Έννοια: Ternary και endless method definition

Πριν αρχίσεις: 005, 004

Η τριαδική έκφραση `condition ? yes : no` ταιριάζει σε δύο σύντομες επιλογές. Ο endless ορισμός `def method(args) = expression` δίνει μέθοδο με μία έκφραση, χωρίς end. Η σύντομη γραφή δεν αλλάζει τη σημασία της επιστροφής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def size_word(n) = n >= 5 ? "large" : "small"
size_word(5)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε paid_label(paid) ως endless method με τριαδική επιλογή: true → "paid", false → "pending". Γράψε και short_total(quantity, price) ως endless method που επιστρέφει γινόμενο.

RUBY
```ruby
paid_label(true)
# => "paid"
```

RUBY
```ruby
paid_label(false)
# => "pending"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 161
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 161 --solution
```

Μια μικρή παραλλαγή

Γράψε endless μέθοδο που μορφοποιεί όνομα μέσα σε [].

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"[Eva]"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

162 / ΜΑΘΗΜΑ

Σύντομες παράμετροι block

Έννοια: Numbered parameters `_1` και implicit `it` στη Ruby 3.4

Πριν αρχίσεις: 014, 013

Σε block χωρίς ρητές παραμέτρους, το _1 αναφέρεται στην πρώτη. Από Ruby 3.4 υπάρχει και το implicit it για μία παράμετρο. Μην ανακατεύεις αυτά τα ονόματα με `|name|` στο ίδιο block. Αν υπάρχει ήδη τοπική it, η αναφορά μπορεί να αφορά εκείνη· προτίμησε σαφή ονόματα σε σύνθετα ή nested blocks.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[[2, 3].map { _1 * 2 }, ["a", "b"].map { it.upcase }]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `implicit_names(names)`. Με δύο map, πρώτα καθάρισε άκρα με _1 και μετά κάνε κεφαλαία με it. Κράτησε κενά αποτελέσματα και αρχικά Strings ίδια. Απαιτεί Ruby 3.4.

```ruby
implicit_names([" Eva ", "Jo"])
# => ["EVA", "JO"]
```

```ruby
implicit_names([" "])
# => [""]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 162
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 162 --solution
```

Μια μικρή παραλλαγή

Με _1 κράτησε μόνο τις θετικές ποσότητες.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

163 / ΜΑΘΗΜΑ

Σε ποια κλήση ανήκει το block;

Έννοια: Προτεραιότητα `{ ... }` και `do ... end`

Πριν αρχίσεις: 106, 107, 027

Οι αγκύλες {} δένονται πιο στενά από το do/end όταν παραλείπεις παρενθέσεις κλήσεων. Έτσι δύο blocks με ίδιο σώμα μπορεί να δοθούν σε διαφορετική μέθοδο. Οι παρενθέσεις κάνουν ρητό ποια κλήση ολοκληρώνεται πριν περάσει ως όρισμα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def binding_outer(value)
  [value, block_given?]
end
def binding_inner
  block_given? ? yield(2) : 2
end
a = binding_outer binding_inner { |n| n * 3 }
b = binding_outer binding_inner do |n| n * 3 end
[a, b]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται wrap_values(values) που ενώνει με ", " μέσα σε []. Γράψε wrapped_doubles(numbers): διπλασίασε με map και do/end και πέρασε το Array στη wrap_values. Χρησιμοποίησε παρενθέσεις ώστε το block να ανήκει στη map.

```ruby
wrapped_doubles([2, 3])
# => "[4, 6]"
```

```ruby
wrapped_doubles([])
# => "[]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 163
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 163 --solution
```

Μια μικρή παραλλαγή

Γράψε ρητά την εσωτερική κλήση με παρενθέσεις σε παράδειγμα δύο μεθόδων.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"[a]"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

164 / ΜΑΘΗΜΑ

Προώθηση ορισμάτων και block

Έννοια: `...` και ανώνυμη προώθηση `*`, `**`, `&`

Πριν αρχίσεις: 118, 114

Το `...` στον ορισμό και στην κλήση προωθεί positional ορίσματα, keywords και block χωρίς να τα ονομάσεις. Οι ανώνυμες `*`, `**`, `&` επιτρέπουν αντίστοιχη προώθηση χωριστά. Η μέθοδος προορισμού συνεχίζει να ελέγχει τη δική της υπογραφή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
def forwarding_target(name, prefix: "")
  text = "#{prefix}#{name}"
  block_given? ? yield(text) : text
end
def forwarding_bridge(...)
  forwarding_target(...)
end
forwarding_bridge("Eva", prefix: "Hi ") { |s| s.upcase }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται base_format(name, prefix: "") που εφαρμόζει και προαιρετικό block στο κείμενο. Γράψε relay_all(...) και relay_parts(*, **, &) που προωθούν στη base_format όλους τους μηχανισμούς ορισμάτων.

```ruby
relay_all("Eva", prefix: "Hi ") { |s| s.upcase }
# => "HI EVA"
```

```ruby
relay_parts("Jo", prefix: "Guest ") { |s| "[#{s}]" }
# => "[Guest Jo]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 164
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 164 --solution
```

Μια μικρή παραλλαγή

Προώθησε μόνο block με ανώνυμο & σε μικρή μέθοδο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `6`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

165 / ΜΑΘΗΜΑ

Μορφοποίηση τιμών

Έννοια: `format` και placeholders

Πριν αρχίσεις: 003, 152

Η `format(pattern, values...)` μορφοποιεί τιμές με placeholders. Το %s δίνει String, %d ακέραιο, %03d ακέραιο με ελάχιστο πλάτος 3 και μηδενικά, %.2f δύο δεκαδικά. Το πλάτος είναι ελάχιστο, δεν κόβει μεγαλύτερες τιμές. Υπάρχουν και ονομασμένα placeholders όπως %s.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
format("%s: %03d", "Seat", 7)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `receipt_line(number, amount)`. Number μη αρνητικός ακέραιος, amount μη αρνητικός αριθμός με έως δύο δεκαδικά. Δώσε "#NNN: AMOUNT", με ελάχιστο τριψήφιο number και ακριβώς δύο δεκαδικά στο amount.

```ruby
receipt_line(7, 2.5)
# => "#007: 2.50"
```

```ruby
receipt_line(1234, 0)
# => "#1234: 0.00"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 165
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 165 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε ονομασμένα placeholders για όνομα και δύο θέσεις.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva: 02"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

166 / ΜΑΘΗΜΑ

Άλλες μορφές literals

Έννοια: Escapes, `%q`, `%Q`, `%w` και heredocs

Πριν αρχίσεις: 003, 027

Το %q δίνει literal String, το %Q επιτρέπει interpolation και escapes και το %w φτιάχνει Array λέξεων. Τα delimiters μπορούν να είναι αγκύλες. Το heredoc `<<~TEXT` δίνει πολυγραμμικό String αφαιρώντας την κοινή αρχική εσοχή. Το τελικό newline παραμένει.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
name = "Eva"
[%q[Hi #{name}], %Q[Hi #{name}], %w[tea cake]]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `invitation(name)`. Με <<~TEXT επίστρεψε δύο γραμμές: "Hello NAME" και " Seats ready" με ακριβώς δύο αρχικά κενά στη δεύτερη. Το αποτέλεσμα έχει τελικό newline. Name είναι String χωρίς newline.

```ruby
invitation("Eva")
# => "Hello Eva\n  Seats ready\n"
```

```ruby
invitation("Jo")
# => "Hello Jo\n  Seats ready\n"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 166
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 166 --solution
```

Μια μικρή παραλλαγή

Με μονά εισαγωγικά στο heredoc delimiter κράτησε interpolation ως κείμενο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Hello \#{name}\n"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

167 / ΜΑΘΗΜΑ

Χαρακτήρας, byte και encoding

Έννοια: UTF-8, `length`, `bytesize`, codepoints και grapheme clusters

Πριν αρχίσεις: 032, 094

Σε UTF-8 String η length μετρά χαρακτήρες κωδικοποίησης, η bytesize bytes και η codepoints δίνει ακέραιους Unicode κωδικούς. Η grapheme_clusters ομαδοποιεί ακολουθίες που λειτουργούν ως ενιαίες μονάδες κειμένου. Ένα γράμμα μαζί με συνδυαστικό τόνο μπορεί να έχει δύο codepoints αλλά ένα grapheme.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
text = "e\u0301"
[text.length, text.bytesize, text.codepoints, text.grapheme_clusters.length]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `text_units(text)`. Δέχεται έγκυρο UTF-8 String. Δώσε Hash {characters: length, bytes: bytesize, codepoints: Array κωδικών, graphemes: πλήθος grapheme clusters}. Μην μεταβάλεις ή κανονικοποιήσεις το κείμενο.

```ruby
text_units("α")
# => {characters: 1, bytes: 2, codepoints: [945], graphemes: 1}
```

```ruby
text_units("e\u0301")
# => {characters: 2, bytes: 3, codepoints: [101, 769], graphemes: 1}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 167
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 167 --solution
```

Μια μικρή παραλλαγή

Χώρισε κείμενο a και e με συνδυαστικό τόνο σε grapheme clusters.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["a", "e\u0301"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

168 / ΜΑΘΗΜΑ

Ταίριασμα δομής πίνακα

Έννοια: `case` / `in`, Array patterns και rest pattern

Πριν αρχίσεις: 120, 008

Το case/in ελέγχει δομή και δεσμεύει ονόματα για τα μέρη που ταιριάζουν. Ένα Array pattern χωρίς rest ζητά ακριβές πλήθος. Το `*names` κρατά τα υπόλοιπα στοιχεία. Οι κυριολεκτικές τιμές μέσα στο pattern, όπως :book, πρέπει να ταιριάζουν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
case [:book, "Eva", 2]
in [:book, name, seats]
  "#{name}: #{seats}"
else
  nil
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `array_command(data)`. Δέχεται Array. Μορφή [:book, name, seats] δίνει {action: :book, name: name, seats: seats}. Μορφή [:batch, *names] δίνει {action: :batch, names: names}. Άλλο σχήμα δίνει nil. Τα δεδομένα των έγκυρων μορφών δίνονται με σωστούς τύπους.

```ruby
array_command([:book, "Eva", 2])
# => {action: :book, name: "Eva", seats: 2}
```

```ruby
array_command([:batch])
# => {action: :batch, names: []}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 168
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 168 --solution
```

Μια μικρή παραλλαγή

Με rest πάρε το πρώτο στοιχείο και όλα τα υπόλοιπα ενός μη κενού Array.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[1, [2, 3]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

169 / ΜΑΘΗΜΑ

Ταίριασμα δομής εγγραφής

Έννοια: Hash patterns και έλεγχος επιπλέον κλειδιών

Πριν αρχίσεις: 168, 037

Το Hash pattern `{name:, seats:}` ζητά παρουσία των κλειδιών και επιτρέπει επιπλέον κλειδιά. Το `**nil` απορρίπτει πρόσθετα πεδία, ενώ `**extra` τα συγκεντρώνει σε Hash. Ένα class pattern όπως String μπορεί να περιορίσει τύπο αντί για συγκεκριμένη τιμή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
case {name: "Eva", seats: 2, note: "x"}
in {name:, seats:, **extra}
  [name, seats, extra]
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `exact_booking(record)`. Με case/in δώσε [name, seats] μόνο για Hash που έχει ακριβώς :name και :seats. Τιμές όπως nil επιτρέπονται σε αυτά τα πεδία. Άλλα σχήματα ή επιπλέον κλειδιά δίνουν nil.

RUBY
```ruby
exact_booking({name: "Eva", seats: 2})
# => ["Eva", 2]
```

RUBY
```ruby
exact_booking({name: nil, seats: 0})
# => [nil, 0]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 169
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 169 --solution
```

Μια μικρή παραλλαγή

Με Hash pattern έλεγξε ότι το name είναι String, επιτρέποντας πρόσθετα πεδία.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `true`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

170 / ΜΑΘΗΜΑ

Περιορισμοί μέσα στο pattern

Έννοια: Pin operator `^`, alternatives και guards

Πριν αρχίσεις: 169, 009

Το `^wanted` χρησιμοποιεί την ήδη υπάρχουσα τιμή αντί να ξαναδεσμεύσει μεταβλητή. Ένα `if` μετά το pattern προσθέτει guard. Η εναλλακτική `A | B` δέχεται ένα από δύο patterns. Ονόματα που δεσμεύονται μόνο σε έναν εναλλακτικό κλάδο έχουν περιορισμούς· κράτα κοινές δεσμεύσεις έξω από την εναλλακτική.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
wanted = "Athens"
case {city: "Athens", seats: 2}
in {city: ^wanted, seats:} if seats > 0
  seats
else
  nil
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `seats_in_city(record, city)`. Οι υπάρχουσες τιμές :city είναι Strings και :seats ακέραιοι. Με pin δώσε seats μόνο όταν το record έχει τα δύο κλειδιά, ίδια city με την παράμετρο και seats > 0. Αλλιώς nil. Πρόσθετα κλειδιά επιτρέπονται.

```ruby
seats_in_city({city: "Athens", seats: 2}, "Athens")
# => 2
```

```ruby
seats_in_city({city: "Patra", seats: 2}, "Athens")
# => nil
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 170
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 170 --solution
```

Μια μικρή παραλλαγή

Δέξου κατάσταση :new ή :queued σε Array δύο στοιχείων και κράτησε το όνομα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

171 / ΜΑΘΗΜΑ

Ανάθεση μέσω pattern

Έννοια: Rightward assignment `=>` και boolean `in`

Πριν αρχίσεις: 169, 082

Η δεξιόστροφη ανάθεση `value => pattern` απαιτεί αντιστοίχιση και προκαλεί NoMatchingPatternError αν δεν γίνει. Το `value in pattern` δίνει boolean. Στην ανάθεση ενός boolean αποτελέσματος γράψε παρενθέσεις, π.χ. `ok = (value in pattern)`.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
data = {name: "Eva", seats: 2}
data => {name:, seats:}
[name, seats]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `required_booking_fields(record)`. Με rightward assignment ζήτησε τα :name και :seats και δώσε [name, seats]. Πρόσθετα πεδία επιτρέπονται. Σε απόν απαιτούμενο πεδίο άφησε NoMatchingPatternError ή υποκλάση του να διαδοθεί.

```ruby
required_booking_fields({name: "Eva", seats: 2, extra: true})
# => ["Eva", 2]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 171
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 171 --solution
```

Μια μικρή παραλλαγή

Διάλεξε boolean έλεγχο όταν μια ασυμφωνία είναι κανονική πιθανότητα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `false`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

172 / ΜΑΘΗΜΑ

Pattern matching δικών σου αντικειμένων

Έννοια: `deconstruct` και `deconstruct_keys`

Πριν αρχίσεις: 068, 168, 169

Τα δικά σου αντικείμενα μπορούν να συμμετέχουν σε patterns. Η deconstruct επιστρέφει Array για array patterns. Η deconstruct_keys(keys) επιστρέφει Hash για hash patterns. Το keys μπορεί να περιέχει ζητούμενα κλειδιά ή nil όταν ζητείται ολόκληρη δομή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class PairPoint
  def deconstruct
    [2, 3]
  end
end
case PairPoint.new
in [x, y]
  x + y
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε Position.new(x, y) με deconstruct → [x, y] και deconstruct_keys(keys) → {x: x, y: y}. Σε αυτή τη μικρή κλάση μπορείς να επιστρέφεις πάντα και τα δύο κλειδιά. Οι συντεταγμένες είναι ακέραιοι.

```ruby
case Position.new(2, 3)
in [x, y]
  [x, y]
end
# => [2, 3]
```

```ruby
case Position.new(4, 5)
in {x:, y:}
  x + y
end
# => 9
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 172
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 172 --solution
```

Μια μικρή παραλλαγή

Όρισε μικρό αντικείμενο που αποσυντίθεται σε Hash με name.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

173 / ΜΑΘΗΜΑ

Μικρός δρομολογητής μηνυμάτων · Εμπέδωση

Έννοια: Σύνθεση parsing και pattern matching

Πριν αρχίσεις: 170, 171, 021

Ένας μικρός router μπορεί να διαλέγει αποτέλεσμα από το σχήμα μηνύματος. Γράψε πρώτα το pattern και μετά την ήδη γνωστή πράξη. Η λογική των κλάδων μένει μία μορφοποίηση ή ένα άθροισμα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
case {type: "greet", name: "Eva"}
in {type: "greet", name:}
  "Hello #{name}"
else
  nil
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `route_message(message)`. Μήνυμα {type: "greet", name: String} δίνει "Hello NAME". Μήνυμα {type: "count", values: Array ακεραίων} δίνει άθροισμα. Άλλες μορφές δίνουν nil. Έλεγξε και τύπους με patterns, επιτρέποντας επιπλέον πεδία.

```ruby
route_message({type: "greet", name: "Eva"})
# => "Hello Eva"
```

```ruby
route_message({type: "count", values: [2, 3]})
# => 5
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 173
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 173 --solution
```

Μια μικρή παραλλαγή

Με pin δέξου μόνο μήνυμα με δοσμένο id.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

174 / ΜΑΘΗΜΑ

Δική σου παραγωγή τιμών

Έννοια: `Enumerator.new` και yielder

Πριν αρχίσεις: 138, 139, 106

Η `Enumerator.new` παίρνει block παραγωγής. Το αντικείμενο yielder δέχεται `<< value` για να δώσει μία τιμή στον καταναλωτή. Το σώμα παραγωγής αρχίζει όταν ζητηθούν στοιχεία. Ο κώδικας μετά από ένα << συνεχίζει όταν προχωρήσει ο καταναλωτής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
e = Enumerator.new do |yielder|
  yielder << "first"
  yielder << "second"
end
[e.next, e.next]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `label_generator(names)`. Επίστρεψε Enumerator.new που για κάθε String name δίνει "Guest NAME" με yielder <<. Κράτησε σειρά και επαναλήψεις. Ο καλών δεν μεταβάλλει names κατά την κατανάλωση.

RUBY
```ruby
label_generator(["Eva", "Jo"]).to_a
# => ["Guest Eva", "Guest Jo"]
```

RUBY
```ruby
label_generator([]).to_a
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 174
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 174 --solution
```

Μια μικρή παραλλαγή

Φτιάξε generator που δίνει αριθμό και έπειτα διπλάσιό του για κάθε είσοδο.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 4, 3, 6]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

175 / ΜΑΘΗΜΑ

Μέθοδος με και χωρίς block

Έννοια: `enum_for` / `to_enum` και `__method__`

Πριν αρχίσεις: 107, 138, 174

Η `enum_for(:method, args...)`, με alias to_enum, φτιάχνει Enumerator που αργότερα θα καλέσει τη μέθοδο με block. Το `__method__` δίνει το τρέχον όνομα μεθόδου ως Symbol. Έτσι μία μέθοδος λειτουργεί και με άμεσο block και χωρίς block.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
def each_tag(names)
  return enum_for(__method__, names) unless block_given?
  names.each { |name| yield("[#{name}]") }
end
each_tag(["A", "B"]).to_a
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `each_positive(values)`. Με block δώσε μία φορά κάθε θετικό ακέραιο του values και επίστρεψε το αρχικό values Array. Χωρίς block δώσε Enumerator μέσω enum_for(**method**, values). Διατήρησε σειρά και μην αλλάξεις values.

RUBY
```ruby
each_positive([-1, 2, 0, 3]).to_a
# => [2, 3]
```

RUBY
```ruby
input = [1, 0, 2]
seen = []
result = each_positive(input) { |n| seen << n }
[seen, result.equal?(input)]
# => [[1, 2], true]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 175
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 175 --solution
```

Μια μικρή παραλλαγή

Πάρε Enumerator από ήδη υπάρχουσα μέθοδο με to_enum.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["a", "b"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

176 / ΜΑΘΗΜΑ

Οι μέθοδοι συλλογών στη δική σου κλάση

Έννοια: `include Enumerable` και πρωτόκολλο `each`

Πριν αρχίσεις: 079, 175

Το module Enumerable δίνει map, select, sum και άλλες πράξεις σε κλάση που παρέχει each. Η each πρέπει να δίνει τα στοιχεία με yield. Χωρίς block μπορεί να επιστρέφει enum_for. Η σειρά που ορίζεις στην each καθορίζει τη σειρά πολλών πράξεων.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class TwoNumbers
  include Enumerable
  def each
    return enum_for(__method__) unless block_given?
    yield 2
    yield 3
    self
  end
end
TwoNumbers.new.sum
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε NumberShelf.new(values) που περιλαμβάνει Enumerable. Η each δίνει όλους τους ακεραίους της δοσμένης λίστας με σειρά και επιστρέφει self. Χωρίς block δίνει Enumerator. Μην γράψεις δικές σου map ή select.

```ruby
s = NumberShelf.new([1, 2, 3])
[s.sum, s.map { |n| n * 2 }, s.select { |n| n > 1 }]
# => [6, [2, 4, 6], [2, 3]]
```

```ruby
NumberShelf.new([4]).each.to_a
# => [4]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 176
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 176 --solution
```

Μια μικρή παραλλαγή

Φτιάξε Enumerable που δίνει δύο Strings και χρησιμοποίησε join μέσω to_a.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva, Jo"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

177 / ΜΑΘΗΜΑ

Υπολογισμός όταν ζητηθεί

Έννοια: `lazy`, `force` και `to_a`

Πριν αρχίσεις: 138, 174

Η `lazy` φτιάχνει Enumerator::Lazy. Οι μετατροπές περιγράφονται τώρα και εκτελούνται όταν ζητηθούν στοιχεία. Η `force` ή η `to_a` υλοποιεί το πεπερασμένο αποτέλεσμα. Μια κανονική map πάνω σε Array υπολογίζει αμέσως όλη την έξοδο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
seen = []
pipeline = [1, 2, 3].lazy.map { |n| seen << n; n * 2 }
before = seen.dup
result = pipeline.take(1).force
[before, result, seen]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `lazy_labels(names, audit)`. Επίστρεψε lazy ακολουθία. Όταν καταναλώνεται όνομα, πρόσθεσέ το στο audit και δώσε κεφαλαίο String. Η δημιουργία της ακολουθίας δεν πρέπει να αλλάζει audit. Ο καλών μπορεί να πάρει μόνο μερικά αποτελέσματα.

RUBY
```ruby
audit = []
e = lazy_labels(["a", "b"], audit)
before = audit.dup
result = e.take(1).force
[before, result, audit]
# => [[], ["A"], ["a"]]
```

RUBY
```ruby
audit = []
result = lazy_labels(["a"], audit).take(0).force
[result, audit]
# => [[], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 177
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 177 --solution
```

Μια μικρή παραλλαγή

Υλοποίησε όλη μια πεπερασμένη lazy ακολουθία με to_a.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 3, 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

178 / ΜΑΘΗΜΑ

Σταμάτημα με αρκετά αποτελέσματα

Έννοια: Lazy φίλτρο και `take`

Πριν αρχίσεις: 177, 015, 132

Το take μετά από φίλτρο ζητά πλήθος αποδεκτών αποτελεσμάτων, όχι πλήθος εξετασμένων εισόδων. Η lazy αλυσίδα προχωρά στην πηγή όσο χρειάζεται και σταματά μόλις συγκεντρώσει τον ζητούμενο αριθμό ή εξαντληθεί.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
seen = []
result = [1, 2, 3, 4, 5].lazy.map { |n| seen << n; n }.select { |n| n >= 3
}.take(2).force
[result, seen]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `first_matching(values, minimum, count, audit)`. Values Array ακεραίων, minimum ακέραιος, count μη αρνητικό. Κατέγραψε στο audit κάθε εξετασμένο στοιχείο. Με lazy select/take δώσε έως count τιμές >= minimum. Σταμάτα μόλις επαρκούν. Μόνο audit αλλάζει.

RUBY
```ruby
audit = []
result = first_matching([1, 3, 2, 4, 9], 3, 2, audit)
[result, audit]
# => [[3, 4], [1, 3, 2, 4]]
```

RUBY
```ruby
audit = []
result = first_matching([1, 2], 3, 2, audit)
[result, audit]
# => [[], [1, 2]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 178
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 178 --solution
```

Μια μικρή παραλλαγή

Πάρε τα πρώτα δύο άρτια από πεπερασμένο lazy Range.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[2, 4]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

179 / ΜΑΘΗΜΑ

Ακολουθία χωρίς έτοιμο τέλος

Έννοια: Endless Range και περιορισμένη κατανάλωση

Πριν αρχίσεις: 028, 178

Το `(start..)` είναι Range χωρίς δεξί όριο. Δεν υλοποιείται ολόκληρο σε Array. Συνδύασε lazy μετατροπές με πεπερασμένο take πριν από force ή to_a, ώστε η κατανάλωση να έχει σαφές τέλος.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
(3..).lazy.map { |n| n * 2 }.take(3).force
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `next_ticket_numbers(start, count)`. Start ακέραιος, count μη αρνητικός ακέραιος. Με endless Range δώσε ακριβώς count διαδοχικούς αριθμούς ξεκινώντας από start. Περιορισμός πριν από υλοποίηση.

```ruby
next_ticket_numbers(7, 3)
# => [7, 8, 9]
```

```ruby
next_ticket_numbers(7, 0)
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 179
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 179 --solution
```

Μια μικρή παραλλαγή

Από θετικούς ακεραίους πάρε τα πρώτα τρία πολλαπλάσια του 5.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[5, 10, 15]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

180 / ΜΑΘΗΜΑ

Σύνδεση και επανάληψη ακολουθιών

Έννοια: `chain` και `cycle`

Πριν αρχίσεις: 138, 177

Η `chain` συνδέει επαναλήψιμες πηγές σε έναν Enumerator. Η `cycle(n)` επαναλαμβάνει μια συλλογή n φορές· χωρίς n μπορεί να συνεχίζει χωρίς τέλος. Για την τελευταία μορφή χρειάζεται όριο κατανάλωσης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["A"].chain(["B", "C"]).to_a
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `repeated_roster(names, rounds)`. Names Array Strings, rounds μη αρνητικός ακέραιος. Με cycle(rounds) δώσε όλες τις επαναλήψεις σε Array. Κράτησε σειρά και μην αλλάξεις names.

```ruby
repeated_roster(["A", "B"], 2)
# => ["A", "B", "A", "B"]
```

```ruby
repeated_roster(["A"], 0)
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 180
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 180 --solution
```

Μια μικρή παραλλαγή

Σύνδεσε μία εισαγωγική τιμή με κύκλο A/B και πάρε μόνο πέντε τιμές.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["start", "A", "B", "A", "B"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

181 / ΜΑΘΗΜΑ

Ομάδες γειτονικών στοιχείων

Έννοια: `chunk` και `chunk_while`

Πριν αρχίσεις: 058, 138

Η `chunk` ομαδοποιεί γειτονικά στοιχεία με ίδιο κλειδί και δίνει [κλειδί, ομάδα]. Η `chunk_while` κρατά συνεχόμενα στοιχεία όσο ο κανόνας για γειτονικό ζευγάρι ισχύει. Η group_by ενώνει όλες τις ίδιες κατηγορίες, ακόμη κι αν βρίσκονται μακριά μεταξύ τους.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
["a", "a", "b", "a"].chunk { |value| value }.to_a
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `consecutive_runs(values)`. Για Array ακεραίων φτιάξε ομάδες όπου κάθε επόμενος αριθμός είναι ο προηγούμενος + 1. Με chunk_while δώσε Array ομάδων. Μην ταξινομήσεις ή αλλάξεις την είσοδο.

RUBY
```ruby
consecutive_runs([1, 2, 5, 6, 2, 3])
# => [[1, 2], [5, 6], [2, 3]]
```

RUBY
```ruby
consecutive_runs([2, 2, 3])
# => [[2], [2, 3]]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 181
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 181 --solution
```

Μια μικρή παραλλαγή

Σύγκρινε πλήθος ομάδων chunk και group_by σε a,a,b,a.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[3, 2]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

182 / ΜΑΘΗΜΑ

Όσο ισχύει ο κανόνας στην αρχή

Έννοια: `take_while` και `drop_while`

Πριν αρχίσεις: 132, 178

Η `take_while` κρατά το αρχικό συνεχόμενο τμήμα όσο ισχύει ο κανόνας. Η `drop_while` παραλείπει αυτό το τμήμα και δίνει όλα τα υπόλοιπα. Μετά την πρώτη αποτυχία του κανόνα δεν αρχίζουν πάλι να φιλτράρουν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = [1, 2, 5, 1]
[values.take_while { |n| n < 3 }, values.drop_while { |n| n < 3 }]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `leading_small(values, limit)`. Για Array ακεραίων και ακέραιο limit δώσε [prefix, rest], όπου prefix το αρχικό συνεχόμενο τμήμα τιμών < limit. Η πρώτη τιμή >= limit και όλα όσα ακολουθούν ανήκουν στο rest.

```ruby
leading_small([1, 2, 5, 1], 3)
# => [[1, 2], [5, 1]]
```

```ruby
leading_small([1, 2], 3)
# => [[1, 2], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 182
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 182 --solution
```

Μια μικρή παραλλαγή

Αφαίρεσε μόνο τα αρχικά κενά Strings, κρατώντας μεταγενέστερα κενά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["A", "", "B"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

183 / ΜΑΘΗΜΑ

Γραμμές και στήλες

Έννοια: `transpose`

Πριν αρχίσεις: 135

Η Array#transpose μετατρέπει γραμμές σε στήλες. Η είσοδος πρέπει να είναι ορθογώνια, με όλες τις εσωτερικές Arrays ίδιου μήκους. Δεν χρειάζεται να γράψεις nested loops για μια έτοιμη αλλαγή διάταξης.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[["A", 2], ["B", 3]].transpose
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `table_columns(rows)`. Η είσοδος είναι Array ισομήκων Arrays. Επίστρεψε τις στήλες με transpose, χωρίς μεταβολή. Για [] δώσε []. Για μη ορθογώνια είσοδο άφησε το IndexError της transpose να διαδοθεί.

```ruby
table_columns([["A", 2], ["B", 3]])
# => [["A", "B"], [2, 3]]
```

```ruby
table_columns([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 183
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 183 --solution
```

Μια μικρή παραλλαγή

Κάνε transpose δύο φορές σε ορθογώνιο πίνακα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[[1, 2], [3, 4]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

184 / ΜΑΘΗΜΑ

Περιστροφή σειράς

Έννοια: `rotate` και `reverse_each`

Πριν αρχίσεις: 053, 138

Η rotate(n) μετακινεί κυκλικά τη σειρά σε νέο Array. Θετικό n μεταφέρει αρχικά στοιχεία στο τέλος, αρνητικό το αντίστροφο. Η reverse_each διασχίζει από το τέλος και χωρίς block δίνει Enumerator. Καμία από τις δύο εκδοχές δεν μεταβάλλει το αρχικό Array.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
["A", "B", "C"].rotate(1)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `rotated_roster(names, offset)`. Δώσε νέο Array ονομάτων με κυκλική μετατόπιση offset ακέραιου. Υποστήριξε μεγαλύτερα offsets, αρνητικά και κενό Array με την έτοιμη rotate.

```ruby
rotated_roster(["A", "B", "C"], 1)
# => ["B", "C", "A"]
```

```ruby
rotated_roster(["A", "B", "C"], -1)
# => ["C", "A", "B"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 184
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 184 --solution
```

Μια μικρή παραλλαγή

Με reverse_each φτιάξε αντίστροφες ετικέτες χωρίς reverse.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["[C]", "[B]", "[A]"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

185 / ΜΑΘΗΜΑ

Ουρά που διαβάζεται σταδιακά · Εμπέδωση

Έννοια: Σύνθεση generator, lazy φίλτρου και παρουσίασης

Πριν αρχίσεις: 174, 177, 178, 088

Σύνδεσε μικρό generator με lazy φίλτρο και όριο. Κατέγραψε ποιες αρχικές γραμμές χρειάστηκε να εξεταστούν. Αυτό κάνει ορατή τη χρονική στιγμή εκτέλεσης χωρίς να χρησιμοποιείς μετρήσεις χρόνου.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
source = Enumerator.new { |y| ["2", "x", "3"].each { |s| y << s } }
source.lazy.filter_map { |s| Integer(s, 10, exception: false) }.take(2).force
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `first_positive_jobs(lines, count, audit)`. Γράψε Enumerator.new που δίνει τις String γραμμές και τις καταγράφει στο audit όταν εξετάζονται. Με lazy μετατροπή, κράτησε θετικούς δεκαδικούς ακεραίους και πάρε έως count. Count μη αρνητικό. Μόνο audit επιτρέπεται να αλλάξει.

RUBY
```ruby
audit = []
result = first_positive_jobs(["x", "0", "2", "3", "9"], 2, audit)
[result, audit]
# => [[2, 3], ["x", "0", "2", "3"]]
```

RUBY
```ruby
audit = []
[first_positive_jobs(["1"], 0, audit), audit]
# => [[], []]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 185
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 185 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε lazy ακολουθία για τις πρώτες δύο μη κενές καθαρές γραμμές.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["A", "B"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

186 / ΜΑΘΗΜΑ

Δικά σου συγκρίσιμα αντικείμενα

Έννοια: `<=>` και `Comparable`

Πριν αρχίσεις: 054, 157, 079

Η `<=>` δίνει αρνητικό, μηδέν ή θετικό αριθμό για μικρότερο, ίσο ή μεγαλύτερο και nil όταν δεν υπάρχει σύγκριση. Με include Comparable αποκτάς <, <=, >, >= και σύγκριση ισότητας βασισμένη σε αυτό το πρωτόκολλο. Μπορείς να συγκρίνεις ήδη έτοιμα Array κλειδιά.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
[[1, "B"] <=> [1, "A"], [1, "A"] <=> [2, "A"]]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε PriorityItem.new(priority, name) με readers. Η σειρά είναι αύξουσα priority και μετά ASCII name. Πρόσθεσε Comparable και <=>. Για άλλο τύπο δώσε nil. Μην γράψεις δικό σου αλγόριθμο sort.

```ruby
items = [PriorityItem.new(2, "A"), PriorityItem.new(1, "B"), PriorityItem.new(1,
"A")]
items.sort.map { |item| [item.priority, item.name] }
# => [[1, "A"], [1, "B"], [2, "A"]]
```

```ruby
PriorityItem.new(1, "A") == PriorityItem.new(1, "A")
# => true
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 186
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 186 --solution
```

Μια μικρή παραλλαγή

Παρατήρησε το πρόσημο της σύγκρισης ακεραίων και την ισότητα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[-1, 1, 0]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

187 / ΜΑΘΗΜΑ

Τελεστές ως μέθοδοι

Έννοια: Ορισμός `+`, `[]` και `[]=`

Πριν αρχίσεις: 041, 068, 070

Πολλοί τελεστές είναι μέθοδοι. Το `a + b` καλεί a.+(b), το object[key] καλεί [] και η ανάθεση object[key] = value καλεί []=. Ο ορισμός τους πρέπει να έχει σαφή σημασία για τον δικό σου τύπο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class SimpleLookup
  def [](key)
    "value for #{key}"
  end
end
SimpleLookup.new[:name]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε SettingsBox.new(values Hash), κρατώντας ρηχό αντίγραφο. Η [] διαβάζει πεδίο, η []= το αλλάζει. Η + δέχεται άλλο SettingsBox και δίνει νέο SettingsBox με δεξιά επικράτηση, χωρίς μεταβολή των δύο αρχικών. Δώσε reader values.

```ruby
s = SettingsBox.new({a: 1})
s[:a] = 2
[s[:a], s[:missing]]
# => [2, nil]
```

```ruby
a = SettingsBox.new({x: 1})
b = SettingsBox.new({x: 2, y: 3})
c = a + b
[c.values, a.values, b.values]
# => [{x: 2, y: 3}, {x: 1}, {x: 2, y: 3}]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 187
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 187 --solution
```

Μια μικρή παραλλαγή

Όρισε + για αντικείμενο που κρατά ακέραιο πλήθος.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

188 / ΜΑΘΗΜΑ

Μέθοδος σε ένα μόνο αντικείμενο

Έννοια: Singleton methods και `class << self`

Πριν αρχίσεις: 073, 160

Η `def object.method` προσθέτει μέθοδο μόνο σε εκείνο το αντικείμενο. Το `class << self` ανοίγει την singleton class του τρέχοντος αντικειμένου. Μέσα σε σώμα κλάσης χρησιμοποιείται για ομαδοποίηση class methods.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
special = "Eva"
def special.badge
  "[#{self}]"
end
special.badge
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε LabelFactory με class method special(name). Κατασκεύασε αντίγραφο του String και όρισε μόνο σε αυτό μέθοδο badge που δίνει "VIP NAME". Επίστρεψε το αντίγραφο. Ομαδοποίησε τη special μέσα σε class << self.

```ruby
LabelFactory.special("Eva").badge
# => "VIP Eva"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 188
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 188 --solution
```

Μια μικρή παραλλαγή

Δώσε δύο class methods μέσω class << self.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["ready", "pending"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

189 / ΜΑΘΗΜΑ

Η σειρά αναζήτησης μεθόδων

Έννοια: `ancestors`, `prepend` και `super`

Πριν αρχίσεις: 078, 079

Η prepend βάζει τις μεθόδους module πριν από την κλάση στην αναζήτηση. Με super συνεχίζεις στην επόμενη υλοποίηση. Η ancestors δείχνει αυτή τη σειρά. Το include συνήθως τοποθετεί το module μετά την ίδια την κλάση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
module BeforeText
  def text
    "[#{super}]"
  end
end
class PlainText
  prepend BeforeText
  def text
    "A"
  end
end
[PlainText.new.text, PlainText.ancestors.first(2)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε CoreFormatter.new(audit) με format(name): πρόσθεσε "core" στο audit και δώσε κεφαλαίο name. Module FormatAudit με prepend προσθέτει "before", καλεί super(name), μετά "after" και επιστρέφει το αποτέλεσμα της γονικής αλυσίδας. Έγκυρα Strings μόνο.

RUBY
```ruby
audit = []
value = CoreFormatter.new(audit).format("Eva")
[value, audit]
# => ["EVA", ["before", "core", "after"]]
```

RUBY
```ruby
CoreFormatter.ancestors.first(2)
# => [FormatAudit, CoreFormatter]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 189
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 189 --solution
```

Μια μικρή παραλλαγή

Σύγκρινε τη θέση module που γίνεται include αντί για prepend.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[IncludingText, IncludedText]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

190 / ΜΑΘΗΜΑ

Επιλογή μεθόδου με όνομα

Έννοια: `respond_to?`, `public_send` και αναζήτηση `method`

Πριν αρχίσεις: 113, 072

Η respond_to? ελέγχει αν ένα αντικείμενο υποστηρίζει μια μέθοδο, με δημόσια ορατότητα από προεπιλογή. Η public_send(name, args...) καλεί δημόσια μέθοδο βάσει ονόματος. Η method(name) δίνει Method αν θέλεις να κρατήσεις την κλήση για αργότερα.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
text = "Eva"
[text.respond_to?(:upcase), text.public_send(:upcase)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `selected_case(text, style)`. Δέχεται String και style Symbol. Επιτρέπονται μόνο :upcase και :downcase. Έλεγξε την επιτρεπτή λίστα και τη respond_to?, κάλεσε με public_send και δώσε String. Άλλο style προκαλεί ArgumentError, "unknown style".

```ruby
selected_case("Eva", :upcase)
# => "EVA"
```

```ruby
selected_case("Eva", :downcase)
# => "eva"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 190
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 190 --solution
```

Μια μικρή παραλλαγή

Πάρε Method από το ίδιο επιλεγμένο όνομα και κάλεσέ το.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"JO"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

191 / ΜΑΘΗΜΑ

Ορισμός μεθόδων από δεδομένα

Έννοια: `define_method`

Πριν αρχίσεις: 109, 069, 190

Η define_method(name) δέχεται block ως σώμα νέας instance method. Το block θυμάται το περιβάλλον του, όπως ένα closure. Είναι χρήσιμη όταν πολλά ονόματα ακολουθούν την ίδια μικρή μορφή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class ColourFlags
  [:red, :blue].each do |colour|
    define_method("#{colour}?") { colour == :red }
  end
end
[ColourFlags.new.red?, ColourFlags.new.blue?]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε StatusFlags.new(status) και δημιούργησε queued?, done?, cancelled? από Array Symbols με define_method. Κάθε μέθοδος συγκρίνει τη δική της κατάσταση με την αποθηκευμένη. Για άγνωστο status όλες δίνουν false.

```ruby
s = StatusFlags.new(:done)
[s.queued?, s.done?, s.cancelled?]
# => [false, true, false]
```

```ruby
s = StatusFlags.new(:other)
[s.queued?, s.done?, s.cancelled?]
# => [false, false, false]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 191
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 191 --solution
```

Μια μικρή παραλλαγή

Δημιούργησε μέθοδο που δέχεται ένα όρισμα μέσω παραμέτρου block.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Hi Eva"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

192 / ΜΑΘΗΜΑ

Μικρές τιμές και δυναμικές κλήσεις · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 157, 186, 190

Συνδύασε ισότητα τιμής, σειρά και δυναμική επιλογή παρουσίασης. Η ίδια μικρή τιμή πρέπει να συμπεριφέρεται συνεπώς σε σύγκριση, Hash και κλήσεις format.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
values = [[2, "A"], [1, "B"], [1, "A"]]
values.sort
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε RankedCode.new(rank, code), με ακέραιο rank και String code. Readers, Comparable με σειρά [rank, code], eql?/hash με την ίδια ταυτότητα. Κράτησε παγωμένο αντίγραφο code. Η format(style) δέχεται :plain ή :upper και καλεί αντίστοιχη δημόσια μέθοδο με public_send. Άλλη επιλογή ArgumentError.

```ruby
a = RankedCode.new(1, "a")
b = RankedCode.new(1, "a")
[a == b, {a => 4}[b]]
# => [true, 4]
```

```ruby
[RankedCode.new(2, "a"), RankedCode.new(1, "b")].sort.map(&:code)
# => ["b", "a"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 192
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 192 --solution
```

Μια μικρή παραλλαγή

Σύγκρινε και αποδιπλοποίησε δύο ίσα έτοιμα Array κλειδιά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[[1, "a"], [2, "a"]]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

193 / ΜΑΘΗΜΑ

Αντιγραφή και αρχικοποίηση αντιγράφου

Έννοια: `clone`, `dup` και `initialize_copy`

Πριν αρχίσεις: 035, 156, 188

Τα dup και clone δημιουργούν ρηχά αντίγραφα, αλλά το clone διατηρεί από προεπιλογή frozen κατάσταση και singleton συμπεριφορά. Η initialize_copy(other) καλείται στο νέο αντίγραφο και επιτρέπει να ορίσεις ποια εσωτερικά αντικείμενα αντιγράφονται. Κάλεσε super για την κανονική αλυσίδα αντιγραφής.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
original = ["A"].freeze
[original.dup.frozen?, original.clone.frozen?]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε TaggedEntry.new(name, tags), με readers. Κράτησε δικό του Array tags. Στη initialize_copy φτιάξε νέο Array tags, ώστε προσθήκη ετικέτας στο dup να μην αλλάζει το αρχικό. Τα ίδια String στοιχεία επιτρέπεται να παραμένουν κοινά.

```ruby
a = TaggedEntry.new("A", ["x"])
b = a.dup
b.tags << "y"
[a.tags, b.tags]
# => [["x"], ["x", "y"]]
```

```ruby
a = TaggedEntry.new("A", ["x"])
b = a.dup
[a.tags.equal?(b.tags), a.tags[0].equal?(b.tags[0])]
# => [false, true]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 193
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 193 --solution
```

Μια μικρή παραλλαγή

Παρατήρησε ότι clone κρατά singleton method ενώ dup όχι.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["VIP", false]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

194 / ΜΑΘΗΜΑ

Ορισμός εναλλακτικού ονόματος

Έννοια: `alias_method`, `alias`, `remove_method` και `undef_method`

Πριν αρχίσεις: 077, 188, 190

Η alias_method(:new_name, :old_name) δίνει δεύτερο όνομα στην τρέχουσα υλοποίηση. Υπάρχει και το keyword alias. Η remove_method αφαιρεί ορισμό μόνο από την τρέχουσα κλάση, επιτρέποντας αναζήτηση σε γονέα. Η undef_method εμποδίζει τη συγκεκριμένη μέθοδο και στην κληρονομική αναζήτηση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class AliasDemo
  def render
    "A"
  end
  alias_method :text, :render
end
AliasDemo.new.text
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται LegacyLabel.new(name) με render → "[NAME]". Πρόσθεσε text ως alias_method και title με keyword alias. Και τα τρία ονόματα πρέπει να δίνουν την ίδια συμπεριφορά χωρίς αντίγραφα σώματος μεθόδου.

```ruby
label = LegacyLabel.new("Eva")
[label.render, label.text, label.title]
# => ["[Eva]", "[Eva]", "[Eva]"]
```

```ruby
LegacyLabel.new("Jo").text
# => "[Jo]"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 194
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 194 --solution
```

Μια μικρή παραλλαγή

Σε δύο υποκλάσεις σύγκρινε remove_method και undef_method πάνω στη label.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["base", false]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

195 / ΜΑΘΗΜΑ

Απουσία μεθόδου ως πρωτόκολλο

Έννοια: `method_missing` και `respond_to_missing?`

Πριν αρχίσεις: 118, 190

Η method_missing καλείται όταν δεν βρεθεί μέθοδος. Αν αναγνωρίζεις συγκεκριμένο δυναμικό όνομα, μπορείς να το χειριστείς. Για όλα τα υπόλοιπα κάλεσε super. Η respond_to_missing? πρέπει να συμφωνεί με τη δυναμική συμπεριφορά, ώστε respond_to? και method να την καταλαβαίνουν.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class MissingDemo
  def method_missing(name, *args, &block)
    return "Eva" if name == :name && args.empty? && !block
    super
  end
  def respond_to_missing?(name, include_private = false)
    name == :name || super
  end
end
[MissingDemo.new.name, MissingDemo.new.respond_to?(:name)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε DynamicRecord.new(fields). Κράτησε μόνο :name και :seats από το Hash. Αν το αντίστοιχο κλειδί υπάρχει, υποστήριξε κλήση χωρίς ορίσματα και block ως ανάγνωση, ακόμη και για nil/false. Άγνωστη μέθοδος ή κλήση με ορίσματα περνά στο super. Πρόσθεσε respond_to_missing?.

RUBY
```ruby
r = DynamicRecord.new({name: nil, seats: false})
[r.name, r.seats, r.respond_to?(:name)]
# => [nil, false, true]
```

RUBY
```ruby
DynamicRecord.new({name: "Eva"}).method(:name).call
# => "Eva"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 195
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 195 --solution
```

Μια μικρή παραλλαγή

Δείξε ότι ένα άγνωστο όνομα παραμένει NoMethodError όταν γίνεται super.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"unknown method"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

196 / ΜΑΘΗΜΑ

Διαφορετικό self μέσα στο block

Έννοια: `instance_exec` και `class_exec`

Πριν αρχίσεις: 108, 109, 191

Η instance_exec εκτελεί block με νέο self και με όσα ορίσματα δώσεις. Οι captured τοπικές μεταβλητές παραμένουν από το περιβάλλον δημιουργίας. Η class_exec κάνει αντίστοιχη εργασία πάνω σε κλάση και επιτρέπει δηλώσεις μεθόδων. Η Class.new δημιουργεί ανώνυμη κλάση που μπορείς να κρατήσεις σε μεταβλητή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
"Eva".instance_exec("Hi") { |prefix| "#{prefix} #{upcase}" }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `apply_object_rule(object, value, &rule)`. Εκτέλεσε rule με self το δοσμένο object και ένα όρισμα value. Επίστρεψε ακριβώς την επιστροφή του block, μαζί με nil/false. Το block δίνεται πάντα. Μην αλλάζεις εσύ το object.

```ruby
apply_object_rule("Eva", "Hi") { |prefix| "#{prefix} #{upcase}" }
# => "Hi EVA"
```

```ruby
suffix = "!"
apply_object_rule("Jo", "Guest") { |prefix| "#{prefix} #{self}#{suffix}" }
# => "Guest Jo!"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 196
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 196 --solution
```

Μια μικρή παραλλαγή

Με class_exec πρόσθεσε reader/writer από όνομα πεδίου σε ανώνυμη κλάση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"Eva"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

197 / ΜΑΘΗΜΑ

Μικρή εσωτερική γλώσσα

Έννοια: DSL με ήδη γνωστά blocks και μεθόδους

Πριν αρχίσεις: 196, 108, 121

Ένα μικρό εσωτερικό DSL μπορεί να είναι συνηθισμένες Ruby κλήσεις με blocks. Η build αλλάζει το self στο αντικείμενο ορισμού, ώστε να γράφεις column(...) απευθείας. Κάθε block στήλης αποθηκεύεται και εκτελείται αργότερα πάνω στην εγγραφή.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
class TinyDefinition
  attr_reader :names
  def initialize
    @names = []
  end
  def column(name)
    @names << name
  end
end
d = TinyDefinition.new
d.instance_exec { column("Name"); column("Seats") }
d.names
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε ReportDefinition.build(&block) που δημιουργεί αντικείμενο και εκτελεί block στο πλαίσιό του. Η column(title, &formatter) αποθηκεύει τίτλο και callable με σειρά δήλωσης. Η render(record) δίνει Array [τίτλος, formatter.call(record)]. Κάθε column δέχεται block. Χωρίς στήλες δώσε [].

RUBY
```ruby
definition = ReportDefinition.build do
  column("Name") { |r| r[:name] }
  column("Seats") { |r| r[:seats] }
  column("Label") { |r| "#{r[:name]}: #{r[:seats]}" }
end
definition.render({name: "Eva", seats: 2})
# => [["Name", "Eva"], ["Seats", 2], ["Label", "Eva: 2"]]
```

RUBY
```ruby
ReportDefinition.build {}.render({})
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 197
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 197 --solution
```

Μια μικρή παραλλαγή

Χτίσε μία μικρή λίστα τίτλων με block που δέχεται ρητό receiver αντί να αλλάζει self.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["Name", "Seats"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

198 / ΜΑΘΗΜΑ

Μικρό API αναφορών · Εμπέδωση

Έννοια: Σύνθεση αντικειμένων, callable και δυναμικού ορισμού

Πριν αρχίσεις: 191, 196, 197

Μπορείς να δημιουργήσεις μια μικρή ανώνυμη κλάση από δοσμένα callable πεδία. Κάθε δυναμική μέθοδος έχει καθαρή υπογραφή και οι επιτρεπτές ονομασίες παραμένουν συγκεκριμένες. Το Class.new αποφεύγει συγκρούσεις μεταξύ διαφορετικών ορισμών.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
klass = Class.new
klass.define_method(:label) { |record| "[#{record[:name]}]" }
klass.new.label({name: "Eva"})
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `column_object(formatters)`. Formatters είναι Hash με Symbol κλειδιά μόνο :name, :seats ή :label και callable τιμές ενός ορίσματος record. Επίστρεψε instance νέας ανώνυμης κλάσης με μία μέθοδο ανά κλειδί μέσω define_method. Η κάθε μέθοδος καλεί τον δικό της formatter. Άγνωστο κλειδί ArgumentError.

RUBY
```ruby
object = column_object({name: ->(r) { r[:name].upcase }, seats: ->(r) { r[:seats]
}})
record = {name: "Eva", seats: 2}
[object.name(record), object.seats(record)]
# => ["EVA", 2]
```

RUBY
```ruby
a = column_object({label: ->(r) { "A" }})
b = column_object({label: ->(r) { "B" }})
[a.label({}), b.label({})]
# => ["A", "B"]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 198
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 198 --solution
```

Μια μικρή παραλλαγή

Πρόσθεσε δύο δυναμικές constant-return μεθόδους σε νέα κλάση.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["L", "R"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

199 / ΜΑΘΗΜΑ

Αλλαγή συμπεριφοράς σε περιορισμένο scope

Έννοια: Refinements, `refine` και `using`

Πριν αρχίσεις: 074, 078, 189

Ένα refinement ορίζεται με refine μέσα σε module και ενεργοποιείται με using σε συγκεκριμένο λεκτικό scope. Δεν αλλάζει καθολικά την κλάση. Το πού γράφτηκε η κλήση μεθόδου καθορίζει αν ισχύει το refinement, όχι απλώς ποιος την κάλεσε αργότερα. Το using δεν μπαίνει μέσα σε μέθοδο.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
class DemoRaw
  def text
    "plain"
  end
end
module DemoRefinement
  refine DemoRaw do
    def text
      "[#{super}]"
    end
  end
end
module DemoRefinedScope
  using DemoRefinement
  def self.render(object)
    object.text
  end
end
[DemoRaw.new.text, DemoRefinedScope.render(DemoRaw.new)]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Δίνεται RawLabel#text → "plain". Γράψε DecoratedLabel refinement που επιστρέφει "[plain]" χρησιμοποιώντας super. Ενεργοποίησέ το μόνο στο FancyScope, του οποίου η self.render(label) καλεί label.text. Εκτός FancyScope η text παραμένει "plain".

```ruby
FancyScope.render(RawLabel.new)
# => "[plain]"
```

```ruby
object = RawLabel.new
FancyScope.render(object)
object.text
# => "plain"
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 199
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 199 --solution
```

Μια μικρή παραλλαγή

Όρισε refinement μιας δικής σου κλάσης που κάνει κεφαλαίο κείμενο μόνο σε ένα module.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["HELLO", "hello"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

200 / ΜΑΘΗΜΑ

Περιορισμένη επανάληψη αποτυχημένης προσπάθειας

Έννοια: `retry` μέσα σε `rescue`

Πριν αρχίσεις: 082, 108, 130

Το retry μέσα σε rescue ξεκινά ξανά το προστατευμένο begin, όχι ολόκληρη τη μέθοδο. Ο μετρητής πρέπει να αρχικοποιείται πριν από το begin. Θέσε ρητό όριο και ξανασήκωσε την τελική εξαίρεση όταν τελειώσουν οι προσπάθειες.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
attempts = 0
begin
  attempts += 1
  raise ArgumentError, "again" if attempts < 2
  attempts
rescue ArgumentError
  retry if attempts < 2
  raise
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `retry_local(max_attempts, &action)`. Max_attempts θετικός ακέραιος. Κάλεσε τη δοσμένη τοπική action μέχρι να πετύχει ή να γίνουν max_attempts. Ξαναδοκίμαζε μόνο ArgumentError. Επίστρεψε την ακριβή επιτυχημένη τιμή. Στο όριο άφησε την τελευταία εξαίρεση να διαδοθεί. Η action στα tests είναι επινοημένη, χωρίς εξωτερικές ενέργειες.

```ruby
calls = 0
result = retry_local(3) { calls += 1; raise ArgumentError if calls == 1; "ready" }
[result, calls]
# => ["ready", 2]
```

```ruby
calls = 0
result = retry_local(3) { calls += 1; false }
[result, calls]
# => [false, 1]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 200
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 200 --solution
```

Μια μικρή παραλλαγή

Κάνε τοπικό retry μέχρι τρίτη προσπάθεια και κράτησε το ίχνος των προσπαθειών.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["try", "try", "try"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

201 / ΜΑΘΗΜΑ

Δύο ανεξάρτητες εργασίες

Έννοια: `Thread` και `join`

Πριν αρχίσεις: 108, 014

Η Thread.new ξεκινά block σε νέο thread. Η join περιμένει την ολοκλήρωσή του. Η value επιστρέφει την τιμή του block και επίσης περιμένει αν χρειάζεται. Η σειρά ολοκλήρωσης μπορεί να διαφέρει από τη σειρά δημιουργίας, γι' αυτό συγκεντρώνουμε αποτελέσματα με ρητή σειρά. Δεν μετράμε επιτάχυνση σε αυτή την άσκηση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
threads = ["a", "b"].map { |name| Thread.new { name.upcase } }
threads.each(&:join)
threads.map(&:value)
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `threaded_labels(names)`. Για μικρό Array Strings φτιάξε ένα thread ανά όνομα, που επιστρέφει "[NAME]" με κεφαλαίο όνομα. Περίμενε όλα με join και δώσε αποτελέσματα στην αρχική σειρά. Κάθε thread δουλεύει ανεξάρτητα, χωρίς κοινό μεταβαλλόμενο αποτέλεσμα.

RUBY
```ruby
threaded_labels(["b", "a", "b"])
# => ["[B]", "[A]", "[B]"]
```

RUBY
```ruby
threaded_labels([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

SH
```sh
bin/lesson 201
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 201 --solution
```

Μια μικρή παραλλαγή

Ξεκίνα δύο διαφορετικές ανεξάρτητες εργασίες και συγκέντρωσε με καθορισμένη σειρά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["a", 5]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

202 / ΜΑΘΗΜΑ

Κοινή μεταβαλλόμενη κατάσταση

Έννοια: `Mutex` και συγχρονισμός

Πριν αρχίσεις: 201, 123

Όταν threads ενημερώνουν κοινή κατάσταση, μια ανάγνωση-αλλαγή-εγγραφή πρέπει να οργανωθεί ως ενιαίο προστατευμένο βήμα. Η Mutex#synchronize επιτρέπει ένα thread τη φορά μέσα στο block. Κρατάμε το προστατευμένο τμήμα μικρό.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
counter = 0
mutex = Mutex.new
threads = Array.new(2) { Thread.new { 3.times { mutex.synchronize { counter += 1 } } } }
threads.each(&:join)
counter
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `synchronized_count(workers, per_worker)`. Workers ακέραιος 0 έως 4 και per_worker ακέραιος 0 έως 100. Ξεκίνα τόσα threads, καθένα αυξάνει κοινό counter per_worker φορές μέσα σε Mutex.synchronize. Περίμενε όλα και επίστρεψε counter.

```ruby
synchronized_count(4, 100)
# => 400
```

```ruby
synchronized_count(0, 5)
# => 0
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 202
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 202 --solution
```

Μια μικρή παραλλαγή

Με την ίδια mutex προστάτευσε προσθήκες δύο threads σε Hash μετρητή.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `{done: 6}`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

203 / ΜΑΘΗΜΑ

Επικοινωνία μεταξύ threads

Έννοια: `Queue` και ολοκλήρωση εργασιών

Πριν αρχίσεις: 201, 130

Η Queue μεταφέρει αντικείμενα μεταξύ threads. Η << προσθέτει και η pop περιμένει όταν η ουρά είναι κενή. Χρειάζεται συμφωνημένο σήμα τέλους. Εδώ οι κανονικές εργασίες είναι Strings, άρα nil μπορεί να σημαίνει ολοκλήρωση.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
queue = Queue.new
worker = Thread.new do
  name = queue.pop
  name.upcase
end
queue << "Eva"
worker.join
worker.value
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `queued_labels(names)`. Names μικρό Array Strings. Ένας καταναλωτής διαβάζει από Queue και δίνει κεφαλαίες τιμές με σειρά. Ο κύριος κώδικας βάζει όλα τα ονόματα και μετά ένα nil. Ο καταναλωτής σταματά στο nil. Περίμενέ τον με join και επίστρεψε το Array του.

RUBY
```ruby
queued_labels(["b", "a", "b"])
# => ["B", "A", "B"]
```

RUBY
```ruby
queued_labels([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 203
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 203 --solution
```

Μια μικρή παραλλαγή

Στείλε δύο ακεραίους και ρητό :done ως σήμα, με άθροισμα στον καταναλωτή.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `5`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

204 / ΜΑΘΗΜΑ

Μικρή ουρά εργασιών με έλεγχο · Εμπέδωση

Έννοια: Σύνθεση concurrency και των προηγούμενων πρωτοκόλλων

Πριν αρχίσεις: 202, 203, 054

Με δύο καταναλωτές η σειρά επεξεργασίας δεν είναι σταθερή. Κράτησε αναγνωριστικό ανά εργασία, συγκέντρωσε αποτελέσματα με mutex και όρισε την τελική σειρά με sort_by. Κάθε καταναλωτής χρειάζεται δικό του σήμα τέλους.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
queue = Queue.new
queue << {id: 2, name: "B"}
queue << {id: 1, name: "A"}
[queue.pop, queue.pop].sort_by { |row| row[:id] }
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `process_jobs(jobs)`. Jobs μικρό Array Hashes με μοναδικό integer id και String name. Με δύο threads και Queue δώσε Array [id, κεφαλαίο name], ταξινομημένο κατά id. Καμία εργασία δεν χάνεται ή επαναλαμβάνεται. Κράτησε εισόδους ίδιες και τερμάτισε και τους δύο workers.

```ruby
process_jobs([{id: 3, name: "a"}, {id: 1, name: "b"}, {id: 2, name: "a"}])
# => [[1, "B"], [2, "A"], [3, "A"]]
```

```ruby
process_jobs([])
# => []
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 204
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 204 --solution
```

Μια μικρή παραλλαγή

Συγκέντρωσε την τιμή επιστροφής δύο threads χωρίς κοινό Array που μεταβάλλεται από αυτά.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[10, 20]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

205 / ΜΑΘΗΜΑ

Έξοδος από εμφωλευμένα blocks

Έννοια: `catch` και `throw`

Πριν αρχίσεις: 022, 129

Το catch(tag) ορίζει σημείο εξόδου και το throw(tag, value) μεταφέρει εκεί τη ροή με τιμή. Μπορεί να φύγει από πολλά nested blocks μαζί. Δεν είναι σύστημα exceptions όπως raise/rescue. Το tag συνδέει το throw με το κοντινότερο αντίστοιχο catch.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
catch(:found) do
  [[1, 2], [3, 4]].each do |group|
    group.each { |n| throw(:found, n) if n > 2 }
  end
  nil
end
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `marker_position(groups, marker)`. Groups Array από Arrays Strings. Βρες την πρώτη εμφάνιση marker με σειρά ομάδων και στοιχείων. Με catch/throw δώσε [δείκτης ομάδας, δείκτης στοιχείου], από 0. Αν λείπει, nil. Η λογική αναζήτησης δίνεται: δύο each_with_index και έλεγχος ==.

```ruby
marker_position([["A"], ["B", "STOP"], ["STOP"]], "STOP")
# => [1, 1]
```

```ruby
marker_position([["A"], []], "STOP")
# => nil
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 205
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 205 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε throw για προκαθορισμένη πρόωρη έξοδο με String αποτέλεσμα.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"done"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

206 / ΜΑΘΗΜΑ

Παύση και συνέχιση εργασίας

Έννοια: `Fiber`, `resume` και `Fiber.yield`

Πριν αρχίσεις: 108, 139

Η Fiber.new δημιουργεί εργασία που ξεκινά με resume. Η Fiber.yield επιστρέφει προσωρινά έλεγχο και τιμή στον καλούντα. Επόμενο resume συνεχίζει από εκεί. Η τελική έκφραση ολοκληρώνει τη Fiber και δίνεται στον καλούντα. Νέο resume μετά την ολοκλήρωση προκαλεί FiberError.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
fiber = Fiber.new do
  Fiber.yield("first")
  "last"
end
[fiber.resume, fiber.resume, fiber.alive?]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `label_fiber(name)`. Επίστρεψε Fiber που στην πρώτη resume δίνει "prepare NAME", στη δεύτερη "ready NAME" και στην τρίτη "done" και ολοκληρώνεται. Χρησιμοποίησε δύο Fiber.yield και τελική έκφραση.

```ruby
f = label_fiber("Eva")
[f.resume, f.resume, f.resume, f.alive?]
# => ["prepare Eva", "ready Eva", "done", false]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 206
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 206 --solution
```

Μια μικρή παραλλαγή

Πέρασε όνομα στη δεύτερη resume και χρησιμοποίησέ το ως επιστροφή της Fiber.yield.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `["name?", "Hi Eva"]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

207 / ΜΑΘΗΜΑ

Πού συνεχίζει η εκτέλεση; · Εμπέδωση

Έννοια: Εμπέδωση μόνο ήδη διδαγμένων εννοιών

Πριν αρχίσεις: 200, 201, 205, 206

Χώρισε τις τέσσερις μορφές ροής: retry επαναλαμβάνει το begin, throw ολοκληρώνει catch, Fiber συνεχίζει από yield και join περιμένει thread. Κατέγραψε αποτελέσματα με καθορισμένη σειρά, χωρίς να μαντεύεις χρονισμό.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

```ruby
fiber = Fiber.new { Fiber.yield("pause"); "finish" }
thread = Thread.new { 7 }
thread.join
[fiber.resume, fiber.resume, thread.value]
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `execution_trace(failures)`. Failures ακέραιος 0 έως 2. Κάνε τοπικές προσπάθειες που σηκώνουν ArgumentError τις πρώτες failures φορές και μετά πετυχαίνουν. Πρόσθεσε catch/throw που δίνει "caught", Fiber που δίνει "pause" και μετά "finish", και Thread που δίνει "thread". Επίστρεψε Hash με attempts, caught, fiber Array και thread τιμή αφού περιμένεις ολοκλήρωση.

```ruby
execution_trace(0)
# => {attempts: 1, caught: "caught", fiber: ["pause", "finish"], thread: "thread"}
```

```ruby
execution_trace(2)
# => {attempts: 3, caught: "caught", fiber: ["pause", "finish"], thread: "thread"}
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 207
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 207 --solution
```

Μια μικρή παραλλαγή

Χρησιμοποίησε catch για να πάρεις την πρώτη τιμή που δίνει μια Fiber.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `"ready"`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

208 / ΜΑΘΗΜΑ

Απομονωμένες εργασίες με μηνύματα

Έννοια: `Ractor` στην έκδοση αναφοράς

Πριν αρχίσεις: 201, 203, 156

Στη Ruby 3.4 η Ractor.new ξεκινά απομονωμένη εργασία. Η send και η Ractor.receive μεταφέρουν μηνύματα, ενώ η take παίρνει έξοδο. Shareable τιμές μπορούν να μοιράζονται· ένα συνηθισμένο μεταβλητό Array αντιγράφεται κατά τη μεταφορά. Με move: true ο αποστολέας χάνει πρόσβαση στο αντικείμενο. Το Ractor API είναι εξειδικευμένο και η άσκηση ελέγχεται στη συγκεκριμένη έκδοση 3.4, όπου εμφανίζεται experimental προειδοποίηση από τη Ruby.

Δες το σε μικρή κλίμακα

Πριν το τρέξεις, πρόβλεψε την τιμή της τελευταίας έκφρασης. Μπορείς να δοκιμάσεις τις γραμμές στο `irb`.

RUBY
```ruby
worker = Ractor.new do
  number = Ractor.receive
  number * 2
end
worker.send(3)
worker.take
```

Η απάντηση και η εξήγηση βρίσκονται στη χωριστή λύση. Το παράδειγμα υπάρχει και στο example.rb.

Η άσκηση

Γράψε `ractor_doubles(numbers)`. Numbers μικρό Array ακεραίων. Ξεκίνα έναν Ractor χωρίς πρόσβαση σε εξωτερικές τοπικές μεταβλητές του block. Στείλε τη λίστα ως μήνυμα, μέσα στον worker δώσε διπλάσιες τιμές με map και πάρε το τελικό Array με take. Μη χρησιμοποιήσεις move στην κύρια άσκηση και κράτησε το αρχικό Array προσβάσιμο και ίδιο.

RUBY
```ruby
ractor_doubles([2, 0, -1])
# => [4, 0, -2]
```

RUBY
```ruby
values = [1, 2]
result = ractor_doubles(values)
[result, values, result.equal?(values)]
# => [[2, 4], [1, 2], false]
```

Γράψε στο exercise.rb. Το περίβλημα δίνεται έτοιμο. Τα tests ελέγχουν την επιστροφή και τις περιπτώσεις της σύμβασης. Οι μη συμπληρωμένες μέθοδοι αποτυγχάνουν εσκεμμένα. Το εργαλείο του μαθήματος πρέπει να φαίνεται και στον κώδικά σου· τα tests ελέγχουν συμπεριφορά.

Από τον φάκελο `syntax-first/ruby`:

```sh
bin/lesson 208
# Μετά την προσπάθειά σου, έλεγχος της έτοιμης λύσης:
bin/lesson 208 --solution
```

Μια μικρή παραλλαγή

Σε δικό σου προσωρινό String σύγκρινε shareable? πριν/μετά το freeze. Έπειτα μετακίνησε άλλο String με move: true και επιβεβαίωσε ότι πρόσβαση στον αποστολέα προκαλεί Ractor::MovedError.

Γράψε την παραλλαγή σε δικό σου πρόχειρο αρχείο. Αναμενόμενο αποτέλεσμα: `[false, true, "HELLO", true]`. Η έτοιμη εκδοχή βρίσκεται στη λύση και στο `variation.rb`.

Διάβασε ξανά την τελευταία έκφραση της μεθόδου ή του block πριν δεις τη λύση. Όταν υπάρχει συλλογή, έλεγξε τι γίνεται στην κενή είσοδο και ποιο αντικείμενο μπορεί να αλλάξει.

Συμπληρωματική αναφορά

Το παράρτημα συμπληρώνει τον χάρτη των 208 μαθημάτων. Περιλαμβάνει μορφές που είναι χρήσιμο να αναγνωρίζεις ή να αναζητάς όταν τις συναντήσεις. Δεν είναι επιπλέον αριθμημένα μαθήματα και δεν προϋποθέτει απομνημόνευση όλων των APIs.

Οι σύντομες σημειώσεις έχουν ως έκδοση αναφοράς τη Ruby 3.4. Η πλήρης προδιαγραφή βρίσκεται στους συνδέσμους κάθε ομάδας.

Literals και τελεστές

Μορφή	Τι να αναγνωρίζεις
`1_000`, `0xff`, `0b1010`, `0o17`	Αριθμητικά literals με διαχωριστικά ή διαφορετική βάση.
`1.25r`, `2i`	Literals Rational και Complex. Οι τύποι αυτοί έχουν δικές τους πράξεις και μετατροπές.
`%W[...]`, `%I[...]`	Οι παραλλαγές των λιστών strings και symbols που επιτρέπουν interpolation.
`%r{...}`, `%s(...)`	Εναλλακτικά literals Regexp και Symbol.
`%x(...)` και backticks	Μορφές εκτέλεσης εντολής· δεν είναι απλά string literals. Η εκτέλεση τοπικού προγράμματος διδάσκεται στο 154.
`(..5)`, `(5..)`, `(...5)`	Beginless και endless ranges. Η χρήση τους εξαρτάται από τον receiver και δεν σημαίνει ότι μπορούν όλα να γίνουν Array.
`**`, `div`, `fdiv`, `divmod`, `%`	Δύναμη και διαφορετικές πράξεις διαίρεσης. Το επιθυμητό είδος αποτελέσματος καθορίζει την επιλογή.
`&`, `|`, `^`, `~`, `<<`, `>>` σε Integer	Πράξεις bits. Ο ίδιος γραπτός τελεστής μπορεί να καλεί διαφορετική μέθοδο σε διαφορετικό receiver.
`and`, `or`, `not`	Λογικοί τελεστές με διαφορετική προτεραιότητα από τα `&&`, `||`, `!`.
`===`	Η σύγκριση που χρησιμοποιείται σε `case` / `when`. Η συμπεριφορά ορίζεται από το αντικείμενο του κανόνα.
`=~`, `!~`	Μορφές σύγκρισης που συναντώνται με regular expressions. Το `match?` παραμένει η σαφής επιλογή για έναν απλό boolean έλεγχο.

Οι μορφές αυτές περιγράφονται στα literals, στην προτεραιότητα τελεστών, στον Integer και στον Regexp.

Scope, κλήσεις και ροή

Μορφή	Τι να αναγνωρίζεις
`{name:}` και `method(name:)`	Συντομευμένη γραφή όταν η τιμή προέρχεται από το αντίστοιχο όνομα. Διάβασέ τη αφού έχουν διδαχθεί Hashes και keywords.
`**nil` σε ορισμό μεθόδου	Ρητή απόρριψη keyword arguments.
`for ... in ...`	Εναλλακτική σύνταξη επανάληψης. Η μεταβλητή της έχει διαφορετικά όρια scope από παράμετρο block της `each`.
`redo`	Ξαναρχίζει την τρέχουσα επανάληψη. Δεν ζητά το επόμενο στοιχείο όπως το `next`.
`begin ... end while ...`	Μορφή επανάληψης που ελέγχει τη συνθήκη μετά την πρώτη εκτέλεση του σώματος.
`..` ή `...` σε κατάλληλη συνθήκη	Η σπάνια μορφή flip-flop. Χρειάζεται ξεχωριστή ανάγνωση του context και δεν είναι εκεί ένα συνηθισμένο Range literal.
`defined?`	Έκφραση που περιγράφει αν και ως τι είναι ορισμένο κάτι. Η επιτυχής απάντηση είναι String, όχι το boolean `true`.
`$name`, `@@name`	Global και class variables. Συγκρίνονται με local, instance και class instance variables ως προς το πού μοιράζεται η κατάσταση.
`__FILE__`, `__dir__`, `__LINE__`	Πληροφορία για το τρέχον πηγαίο αρχείο και τη θέση του κώδικα.
`__method__`, `__callee__`	Πληροφορία για το όνομα της μεθόδου, με διαφορά όταν χρησιμοποιείται alias.
Setter assignment	Η σύνταξη `object.value = x` έχει ως τιμή το `x`, ακόμη κι αν η μέθοδος setter επιστρέφει κάτι διαφορετικό.

Δες assignment, control expressions, methods και calling methods για τις ακριβείς μορφές.

Αντικείμενα και κύκλος ζωής προγράμματος

Μορφή	Τι να αναγνωρίζεις
`to_s` / `to_str`, `to_a` / `to_ary`, `to_i` / `to_int`	Πρωτόκολλα ρητής και έμμεσης μετατροπής. Δεν θεωρούμε τα δύο ονόματα κάθε ζεύγους εναλλάξιμα.
`module_function`	Ένας ακόμη τρόπος να εκτεθούν λειτουργίες module, με συγκεκριμένη σχέση προς τις instance methods του.
`autoload`	Δήλωση αρχείου που θα φορτωθεί όταν ζητηθεί μια σταθερά.
`undef`	Keyword που αφαιρεί τη διαθεσιμότητα μεθόδου· σχετίζεται με την ενότητα δυναμικών ορισμών.
`binding`, `eval`, `instance_eval`, `class_eval`	Εργαλεία του περιβάλλοντος εκτέλεσης και δυναμικής αξιολόγησης. Το scope και η μορφή String ή block χρειάζονται χωριστή μελέτη.
`method_added`, `included`, `inherited`	Hooks για συγκεκριμένα γεγονότα ορισμού μεθόδων, συμπερίληψης module ή δημιουργίας υποκλάσης.
`BEGIN`, `END`, `at_exit`	Ενέργειες σε διαφορετικά σημεία του κύκλου ζωής ενός Ruby προγράμματος.
`__END__`, `DATA`	Τέλος του πηγαίου κώδικα και πρόσβαση στα δεδομένα που ακολουθούν στο κύριο αρχείο.
`# frozen_string_literal: true`	Magic comment που επηρεάζει τα String literals του αρχείου.
`# encoding: UTF-8`	Magic comment για την κωδικοποίηση του πηγαίου αρχείου.

Τα πρωτόκολλα και οι callbacks αναφέρονται στα Object, Module, Class και Kernel. Οι ειδικές μορφές αρχείου βρίσκονται στα miscellaneous syntax και comments.

Συγγενικές λειτουργίες για αναζήτηση

Στα βασικά μαθήματα επιλέγουμε τις λειτουργίες που δείχνουν διαφορετικές ιδέες. Όταν προκύψει συγκεκριμένο αίτημα, αναζητούμε και τις παραλλαγές τους.

Ανάγκη	Συγγενικές λειτουργίες
Πρόσθετη αναζήτηση και φίλτρο	`find_index`, `grep`, `grep_v`, `select` / `filter` / `find_all`.
Περισσότερα χωρίσματα ακολουθίας	`slice_before`, `slice_after`, `slice_when`.
Παραλλαγές κατάταξης	`minmax`, `minmax_by`, επιλογή περισσότερων από ενός μικρότερων ή μεγαλύτερων στοιχείων.
Παραλλαγές μεταβολής Array	`pop`, `shift`, `unshift`, `insert`, `delete_at`, `replace`, `fill`, `flatten`.
Περισσότερες πράξεις Hash	`invert`, `to_h`, `compact`, `merge` με block σύγκρουσης.

Ανάγκη	Συγγενικές λειτουργίες
Περισσότερη επεξεργασία String	`tr`, `delete_prefix`, `delete_suffix`, `unicode_normalize`, `encode`.
Αναπαραγώγιμα τυχαία παραδείγματα	`Random.new` με ρητό seed και οι αντίστοιχες κλήσεις επιλογής τιμών.
Πρόσθετα τοπικά formats και βοηθήματα	`YAML.safe_load`, `URI`, `Tempfile`, `Logger`, `Benchmark`, `pp`.

Αναζήτησε την υπογραφή και τις ιδιαιτερότητες στις αναφορές Enumerable, Array, Hash, String και standard library. Κάθε πρόσθετη άσκηση πρέπει να εξηγεί πρώτα τη συγκεκριμένη νέα συμπεριφορά που χρησιμοποιεί.

Πίσω στη σειρά · Αναλυτικός χάρτης